UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Abner Cardoso da Silva

Análise de um Algoritmo Paralelo de Otimização por Enxame de Partículas Semi-Autônomas

Belém-PA
2019

Abner Cardoso da Silva

Análise de um Algoritmo Paralelo de Otimização por Enxame de Partículas Semi-Autônomas

Trabalho de conclusão de curso submetido à banca da Faculdade de Computação da Universidade Federal do Pará como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Claudomiro de Souza de Sales Júnior

Belém-PA

2019

Abner Cardoso da Silva

Análise de um Algoritmo Paralelo de Otimização por Enxame de Partículas Semi-Autônomas

Trabalho de conclusão de curso submetido à banca da Faculdade de Computação da Universidade Federal do Pará como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dr. Claudomiro de Souza de Sales Júnior
Orientador - PPGCC/UFPA

Prof. Dr. Reginaldo Cordeiro dos Santos Filho
Coorientador - UFPA

Prof. Dr. Filipe de Oliveira Saraiva
Avaliador - PPGCC/UFPA

Prof. Dr. Josivaldo de Souza Araújo
Avaliador - PPGCC/UFPA

Belém-PA
2019

Agradecimentos

Primeiramente, gostaria de agradecer ao meu orientador Claudomiro de Souza de Sales Júnior e meu coorientador Reginaldo Cordeiro dos Santos Filho, por terem me direcionado e tornado esta pesquisa possível.

Agradeço à meus pais Maria Ivaneide Cardoso da Silva e Sinval Oliveira da Silva, que sempre me acolheram de forma plena e tornaram possível a conclusão dessa longa jornada pela graduação.

Agradeço aos meus irmãos, em especial meu irmão mais velho Fabricio Cardoso da Silva e minha irmã mais nova Esther Cardoso da Silva, que sempre estiveram ao meu lado oferecendo o máximo de apoio.

Agradeço à minha companheira Rosa Hiolanda Abreu de Sousa, que me ofereceu todo o suporte possível durante os últimos quatro anos e tornou essa jornada menos árdua.

Por fim, porém não menos importante, agradeço aos meus grandes amigos Bruno Yusuke Kitabayashi e Emanuel Antônio de Melo Mesquita Neto, que foram grandes parceiros e de grande assistência em diversos momentos da graduação.

Resumo

Na área da engenharia é comum o aparecimento de problemas da classe NP-Difícil. Em razão da ambiguidade acerca da existência de algoritmos polinomiais para solucionar esses problemas, são utilizadas técnicas que demandam grande quantidade de recursos computacionais para encontrar respostas viáveis. Dependendo do cenário da aplicação, essas alternativas podem se mostrar impraticáveis devido o excessivo tempo de processamento que exigem. Nesse contexto, são propostas as meta-heurísticas, que se estabelecem como métodos estocásticos para otimização do processos de busca de soluções. Esses métodos são caracterizados pelo seu comportamento estocástico, por serem independentes do problema abordado e, para o caso de problemas não polinomiais, por conseguirem apresentar soluções factíveis com tempos de processamento inferiores aos de métodos exatos. Nessa classe de algoritmos, tem grande destaque o PSO (*Particle Swarm Optimizer*), um algoritmo bioinspirado que visa utilizar modelos abstratos de simulação do comportamento coletivo de animais para otimizar o processo de exploração do espaço de busca de um determinado problema. Esse modelo é notório em função da facilidade de implementação e baixo custo computacional. No entanto, o algoritmo em sua forma mais simples, apresenta certas desvantagens em relação ao modo com que navega o espaço de busca, o que pode influenciar o resultado final. Para tentar amenizar esses problemas, a literatura apresenta uma abundância de variações do PSO com diferentes tipos de operadores. Em trabalhos recentes, uma nova variação denominada SAPSO (*Semi-Autonomous Particle Swarm Optimizer*), que integra operadores de diversidade, cálculo de gradiente e atração e repulsão de partículas, tem apresentado bons resultados em relação a outros algoritmos conhecidos no meio acadêmico. Por se tratar de um trabalho recente, existem poucas pesquisas que explorem o potencial desse algoritmo em diferentes cenários. Tendo isso em mente, este trabalho se propõe a introduzir uma variação do SAPSO em um ambiente de processamento paralelo. Para tal, foi implementado um algoritmo, nomeado PSAPSO (*Parallel Semi-Autonomous Particle Swarm Optimizer*), utilizando a linguagem de programação C++ em conjunto com a API OpenMP. Para avaliar o algoritmo resultante, esse foi submetido a funções de teste que desafiam sua capacidade de exploração em diferentes aspectos. Para os cenários avaliados, os resultados evidenciam um bom ganho de velocidade e uma melhoria na capacidade de convergência do PSAPSO em relação ao SAPSO.

Palavras-chave: Otimização por enxame de partículas, Paralelização, Meta-heurística.

Abstract

In the engineering field, NP-Hard problems are common. Because of the ambiguity about the existence of polynomial-time algorithms to solve these problems, techniques that require a great amount of computational resources are used to find practicable solutions. Depending on the application scenario, these alternatives may be impractical due to the excessive processing time they require. In this context, meta-heuristics are proposed, which are established as stochastic methods to optimize solution search processes. These methods are characterized by their stochastic behavior, because they are independent of the problem addressed and, in the case of non-polynomial problems, they can present feasible solutions with lower processing times than known solutions. In this class of algorithms the PSO (Particle Swarm Optimizer) stands out, which is a bioinspired algorithm that aims to use abstract models of simulation of the collective behavior of animals to optimize the process of exploring the search space of a given problem. This model is notorious for its ease of implementation and low computational cost. However, this algorithm, in its simplest form, has certain disadvantages in relation to the way it browses the search space, which can influence the final result. To try to mitigate these problems, the literature presents an abundance of variations of the PSO with different types of operators. In recent works, a new variation called SAPSO (Semi-Autonomous Particle Swarm Optimizer), which integrates operators of diversity, gradient calculation and attraction and repulsion of particles, has presented good results in relation to other algorithms known in the academic world. Because it is a recent work, there is little research that explores the potential of this algorithm in different scenarios. With this in mind, this paper proposes to introduce a variation of SAPSO in a parallel processing environment. For this, an algorithm, named PSAPSO (Parallel Semi-Autonomous Particle Swarm Optimizer), was implemented using the C++ programming language combined with the OpenMP API. In order to evaluate the resulting algorithm, it has been subjected to test functions that challenge its exploration capacity in different aspects. In the proposed scenarios, the results show improvements in processing speed and convergence capability of PSAPSO in relation to SAPSO.

Keywords: Particle Swarm Optimizer, Parallelization, Meta-heuristics.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

UFPA	Universidade Federal do Pará
PPGCC	Programa de Pós-Graduação em Ciência da Computação
PSO	*Particle swarm optimization*
API	*Application Program Interface*
EPSO	*Evolutionary Particle Swarm Optimization*
DPSO	*Discrete Particle Swarm Optimization*
SAPSO	*Semi-Autonomous Particle Swarm Optimization*
PSAPSO	*Parallel Semi Autonomous Particle Swarm Optimization*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
PCLPSO	*Parallel Comprehensive Learning Particle Swarm Optimization*
GIS	*Greedy Information Swap*
GC-PSO	*Guaranteeded Convergence Particle Swarm Optimization*
SPSO-2011	*Standard Particle Swarm Optimization*
MPI	*Message Passing Interface*
ARPSO	*Attraction and Repulsion Particle Swarm Optimization*
GPSO	*Gradient-based Particle Swarm Optimization*
DGHPSOGS	*Diversity-guided Hybrid PSO based on Gradient Search*
PCAM	*Partitioning Communication Agglomeration Mapping*
GCC	*GNU Compiler Collection*
GDB	*GNU Debugger*
IDE	*Integrated Development Environment*
SMT	*Simultaneous Multithreading*
AMD	*Advanced Micro Devices, Inc.*

Sumário

1 Introdução

Em diversas áreas de estudo são recorrentes os problemas de otimização classificados como NP-Difíceis, considerados intratáveis pois não há conhecimento de algoritmos determinísticos que sejam executadas em tempo polinomial. Esta classe de problemas demanda grande poder computacional, uma vez que, para garantir soluções ótimas normalmente exigem a avaliação de todos os resultados possíveis, o que os torna computacionalmente inviáveis. Portanto, como forma de amenizar estes problemas, são aplicados algoritmos de busca estocástica, que visam avaliar o espaço de soluções de modo inteligente, como otimização por enxame de partículas (*Particle Swarm Optimization* – PSO), algoritmo genético (*Genetic Algorithm*) ou recozimento simulado (*Simulated Annealing*).

Dentre os inúmeros métodos de busca estocástica, destaca-se o algoritmo de PSO (KENNEDY; EBERHART, 1995), que trata de um método bioinspirado de busca global, idealizado a partir da análise do comportamento social de coletivos de animais como cardumes e revoadas de pássaros, em que a inteligência dos indivíduos que compõe um conjunto é a chave fundamental para guiar a população a um objetivo em comum. Originalmente proposto como um método com baixo custo computacional, o algoritmo apresenta simplicidade das operações matemáticas e baixos requisitos de uso de memória, mas também possui forte sensibilidade a definição dos parâmetros iniciais de execução.

Em sua versão canônica, o PSO apresenta algumas limitações intrínsecas que afetam diretamente a eficiência de seus resultados, como a utilização de determinados parâmetros, que pode enviesar os resultados da execução. Dessa forma, várias pesquisas foram realizadas com o intuito de modelar operadores que amenizem os problemas causados por tais limitações, facilitando a fuga de ótimos locais e a convergência para o melhor global (GBENGA; RAMLAN, 2016).

Algumas modificações com contribuições notáveis são: *Evolutionary* PSO (EPSO), que busca unir propriedades de algoritmos evolutivos e de inteligência de enxame (MIRANDA; FONSECA, 2002); *Discrete* PSO (DPSO), criado com o propósito de examinar espaços de busca discretos (KENNEDY; EBERHART, 1997); e o PSO com coeficiente de constrição, que adiciona um novo fator responsável por balancear os processos de *exploration* e *exploitation* (CLERC; KENNEDY, 2002).

Outro algoritmo com relevância, recentemente publicado, é o *Semi-Autonomous* PSO (SAPSO), que busca melhorar as qualidades do PSO, integrando diferentes tipos de operadores (SANTOS et al., 2018). Esse apresenta melhoramentos dos mecanismos de exploração do espaço de busca e controle da população, a partir do uso de informações de gradiente e controla de diversidade dos indivíduos utilizando um sistema de atração e

repulsão.

Abordagens heurísticas podem resultar em grandes tempos de processamento, em função da grande complexidade de problemas reais. Tendo em vista essa dificuldade e a proliferação de processadores e processadores gráficos com grande número de núcleos, tornou-se inevitável a integração de PSO a modelos paralelos para distribuir a carga de processamento, com o objetivo de reduzir seus tempos de execução e melhorar a qualidade das soluções a partir da aplicação de modelos de comunicação (DALI; BOUAMAMA, 2018).

Este trabalho trata da aplicação de um operador paralelo ao SAPSO. A versão paralela possibilita a execução concorrente de múltiplas instâncias do algoritmo SAPSO, que se comunicam entre si compartilhando partículas consideradas as mais aptas dentre todos os enxames. Para distinguir o algoritmo paralelo de sua versão sequencial, este foi nomeado *Parallel Semi Autonomous* PSO (PSAPSO).

1.1 Contextualização

A otimização de processos é um desafio enfrentado em diferentes áreas do conhecimento, onde, muitas vezes, modelos de busca determinística, que garantam encontrar soluções ótimas, não são aplicáveis à resolução dos problemas enfrentados. Nesses casos, para evitar a necessidade de verificar todas as possíveis soluções com o objetivo de encontrar soluções ótimas, são apresentadas as meta-heurísticas, que podem ser definidas como métodos estocásticos que exploram o espaço de busca de um determinado problema, a fim de reduzir os custos computacionais necessários para encontrar respostas factíveis.

Inteligência de enxame é um subconjunto de algoritmos de inteligência artificial que comporta métodos que buscam soluções a partir da simulação do comportamento coletivo de sistemas naturais ou artificiais, como coletivos de animais (LI; CLERC, 2019). No ramo da inteligência de enxame, o algoritmo de PSO tem ganhado notoriedade devido a sua grande flexibilidade e requerer apenas um conjunto de operações simples, porém o algoritmo originalmente proposto por Kennedy e Eberhart (KENNEDY; EBERHART, 1995), apresenta características de implementação que podem afetar negativamente o resultado final (GBENGA; RAMLAN, 2016).

Ao longo dos anos foram propostas diferentes variações do algoritmo com o intuito de amenizar os fatores negativos da versão canônica, tais modificações podem ser classificadas como: métodos de aperfeiçoamento da inicialização da população (GAO; LIU; HUANG, 2012); desenvolvimento de mecanismos topológicos, que definem estruturas de comunicação entre os indivíduos (MENDES; J.KENNEDY; NEVES, 2004); criação de novos parâmetros (CLERC; KENNEDY, 2002) e algoritmos híbridos que buscam mesclar o PSO com demais meta-heurísticas (TIAN; SHI, 2018), essas foram de grande importância para o

estabelecimento do conhecimento atual acerca da área.

1.2 Trabalhos Correlatos

Nos anos subsequentes à primeira publicação acerca do PSO, vários trabalhos foram publicados com interesse em analisar suas qualidades e deficiências (SHI; EBERHART, 1999; OZCAN; MOHAN, 1999). Dentre os trabalhos publicados, destacam-se as propostas de criação de modelos de múltiplos enxames, como (BASKAR; SUGANTHAN, 2004), onde é apresentado um modelo simples de divisão da população em subgrupos, com mecanismo de troca dos melhores indivíduos. A partir destas propostas e da necessidade de atender às demandas de recursos, diretamente relacionadas ao nível de complexidade dos problemas abordados, foi proposta a utilização de sistemas paralelos. Em (CHANG; CHU; RODDICK, 2005) são usadas estratégias de paralelização na CPU, aplicadas ao PSO canônico, onde os autores se aprofundam em métodos de troca de informações entre os enxames. Trabalhos atuais demonstram uma tendência para a paralelização em GPU (WACHOWIAK; TIMSON; DUVAL, 2017), uma vez que esta possui uma arquitetura própria para execução de processos paralelos e um número maior de núcleos quando comparada à CPU.

Em (GüLCü; KODAZ, 2015) é apresentado o *Parallel Comprehensive Learning* PSO (PCLPSO), que utiliza uma estratégia de cooperação entre enxames denominada *Greedy Information Swap* (GIS), onde são definidos quatro fatores fundamentais para garantir bons resultados, sendo estes: período de migração, topologia de troca de informações, o tipo de informação enviada e a política de integração. Os autores demonstram a eficiência do modelo GIS, em relação à implementação sequencial, a partir de análises de *speedup* e comparações de resultados.

O trabalho publicado por Szczepanski, Erwinski e Paprocki (SZCZEPANSKI; ERWINSKI; PAPROCKI, 2017) aborda a paralelização em CPU da variação *Guaranteeded Convergence* PSO (GC-PSO), aplicada a otimização do processo de criação de perfis de alimentação para sistemas de controle numérico computadorizado. Este busca afirmar a importância da paralelização para a redução do tempo de execução, porém o trabalho possui uma análise escassa dos resultados de *speedup* em relação ao número de núcleos utilizados.

Abdullah, Saleh e Saif (ABDULLAH; SALEH; SAIF, 2018) informam a importância da utilização de modelos paralelos em relação ao desempenho, tomando como base os resultados obtidos a partir da utilização de um esquema de paralelização em ilhas, implementado em CPU. Neste é feita uma análise de *speedup* e qualidade da solução, demonstrando suas vantagens em relação a sua versão serial e a outra variação chamada de *Standard PSO* (SPSO-2011). No entanto, o trabalho faz uma análise puramente de de-

sempenho em relação a uma variação específica, não apresentando mecanismos de controle do enxame que evitem a ocorrência de situações adversar como estagnação em ótimos locais ou explosão de velocidade.

Cicirelli et al. (CICIRELLI et al., 2015) aborda estratégias de paralelização em sistemas de otimização por inteligência de enxames a partir de duas perspectivas diferentes, sendo elas: particionamento de espaço, em que o próprio espaço de busca é subdividido entre todas as unidades de processamento ativas; e particionamento de dados, onde as próprias partículas são distribuídas entre as unidades de processamento. O trabalho demonstra que ambos os modelos são eficientes e escaláveis, no entanto particionamento de espaço influencia apenas no tempo de execução do algoritmo, resultando em soluções semelhantes às de um ambiente sequencial, enquanto que particionamento de dados garante uma melhoria na qualidade dos resultados.

Em (WANG; SU; ZHANG, 2017) é exposto a utilização de interface de passagem de mensagem (*Message Passing Interface* – MPI) para a execução concorrente do algoritmo de PSO com ajuste dinâmico do peso inercial, aplicado à redução de custos em sistemas de gestão de micro-redes. A análise feita pelos autores têm grande foco no efeito da modificação do tamanho da população sobre a eficiência do algoritmo, não apresentando uma análise mais aprofundada acerca da eficiência de processamento no ambiente paralelo ou de modelos de comunicações entre enxames.

Santos et al. (SANTOS et al., 2018) apresenta o SAPSO, que tem como objetivo reduzir esforços computacionais necessários para exploração local a partir da utilização da informação de gradiente, e auxilia na fuga de ótimos locais e manutenção da pluralidade das soluções a partir do controle de diversidade associado a um sistema de atração e repulsão. O trabalho toma como inspiração as pesquisas de (VESTERSTRØM; RIGET, 2002) que introduz o *Attraction and Repulsion* PSO (ARPSO), que estabelece o modelo de atração e repulsão; (NOEL, 2012) que divulga o *Gradient-based* PSO (GPSO), que utiliza informações de gradiente para buscas locais; e (HAN; LIU, 2014) onde é apresentado o *Diversity-guided Hybrid* PSO *based on Gradient Search* (DGHPSOGS), que faz uso de informações do gradiente e controle de diversidade.

1.3 Motivação

O aperfeiçoamento do algoritmo canônico é um tema que vem ganhando forte repercussão no ambiente acadêmico, com pesquisas que propõem métodos para solucionar deficiências presentes no algoritmo de PSO, embora em muitos casos tais trabalhos são apresentados como novidades, tratam-se de conceitos já conhecidos na área de otimização (SÖRENSEN, 2015). Portanto, é de grande interesse para pesquisadores da área, trabalhos que sejam capazes de analisar e sintetizar de modo coerente, informações de modelos e

métodos inovadores.

Outro fator de interesse é a grande demanda de processamento necessário na otimização de problemas complexos. Muitos trabalhos da literatura buscam, na computação paralela, soluções de alta desempenho com baixos tempos de processamento para PSO. Além do mais, por possibilitar a execução de múltiplas instâncias de um mesmo algoritmo simultaneamente, este paradigma permite criar operadores que proporcionem a troca de informação entre os enxames (ABDULLAH; SALEH; SAIF, 2018).

Este trabalho almeja contribuir para pesquisas acerca de uma nova variação do algoritmo de PSO, que devido se tratar de um trabalho recente, carece de estudos que avaliem seu comportamento quando submetido a diferentes paradigmas computacionais. Logo, o desenvolvimento de uma pesquisa que detalhe o desempenho do algoritmo em ambiente de processamento paralelo, torna-se um ponto de grande interesse para a comunidade científica.

1.4 Justificativa

Em virtude dos resultados promissores apresentados e obtidos pelo SAPSO (SANTOS et al., 2018) e da escassez de trabalhos que se aprofundem na proposta apresentada em função de ser um método recente, é de grande relevância trabalhos, como o presente estudo, que se propõem a expandir o escopo de pesquisa acerca deste novo método.

Outro ponto relevante são as fragilidades presentes no SAPSO. Para garantir o *exploitation* este faz uso do cálculo do gradiente, que é uma operação computacionalmente custosa. Além disso, este possui um mecanismo simplório para verificação de estagnação das partículas em um mínimo local. Tais desvantagens podem impactar nos resultados e no tempo de execução, portanto a utilização de modelos paralelos pode amenizar seus efeitos.

Tendo conhecimento de que o desempenho é um fator crítico em métodos de otimização e que a distribuição de processamento é um procedimento comum para melhoramento de desempenho, jugou-se apropriado dar início a pesquisas que abordem o comportamento deste novo método em um ambiente paralelo. Tomando como fundamento o algoritmo SAPSO e suas melhorias em relação aos demais algoritmos encontrados na literatura (SANTOS et al., 2018), este trabalho busca contribuir com a análise desse algoritmo, implementado de modo concorrente, provendo uma comparação do seu desempenho em relação a sua versão sequencial e, por fim, analisar estatisticamente os resultados obtidos. Desse modo esta pesquisa visa se fundamentar como alicerce para possíveis trabalhos futuros que almejem avaliar de modo mais detalhado modelos paralelos do algoritmo SAPSO.

O modelo proposto busca melhorar o desempenho de sua versão sequencial a partir da execução síncrona do algoritmo em múltiplas *threads*, possibilitando aumentar o número de partículas responsáveis por explorar o espaço de busca, sem que hajam grandes impactos no tempo de execução do mesmo. Para que seja possível melhorar as soluções encontradas, a partir da colaboração entre os enxames, é adotado um paradigma de paralelização mestre-escravo, em que uma *thread*, eleita como mestre, analisa e compartilha as informações das melhores partículas encontradas entre todos os enxames.

1.5 Objetivos

A pesquisa desenvolvida em (SANTOS et al., 2018) acerca da versão sequencial do algoritmo SAPSO, o compararam com os algoritmos ARPSO, DGHPSOGS e GPSO, demonstrando vantagens no que diz respeito a agilidade com que o algoritmo converge para o melhor global e a qualidade da solução encontrada. Portanto, este trabalho parte do pressuposto de que a implementação sequencial do algoritmo SAPSO foi comprovado ser mais eficiente dentre diferentes variações do PSO. Dessa forma, a pesquisa tem foco na análise do desempenho do PSAPSO em relação a sua versão sequencial. Como objetivos específicos deste trabalho podem ser citados:

- Paralelizar o algoritmo SAPSO, utilizando o paradigma mestre-escravo para possibilitar a troca de informação entre enxames.
- Avaliar os resultados obtidos no PSAPSO e compará-los com os do SAPSO.
- Avaliar a eficiência a partir da análise do *speedup* do PSAPSO em relação ao SAPSO.
- Analisar estatisticamente os resultados obtidos nas execuções do PSAPSO, a fim de formalizar suas possíveis vantagens.

1.6 Metodologia

O SAPSO e o PSAPSO foram submetidos a testes com os problemas apresentados por De Jong (JONG, 1975) em sua pesquisa, que conta com um conjunto de funções comumente utilizadas para avaliação de meta-heurísticas, por conter variados tipos de funções com comportamentos diferentes que apresentam grandes desafios para encontrar o ótimo global, mostrando-se um meio eficiente de demonstrar as capacidades do modelo estudado sem a necessidade de aplicação em um problema real. Descrições mais detalhadas das funções utilizadas serão apresentadas ao decorrer deste trabalho.

Para a avaliação do desempenho do algoritmo estudado foram vistos como importantes os seguintes fatores: o número de iterações necessárias para encontrar a solução,

o tempo de execução, acurácia das soluções e o *speedup* em relação a versão sequencial. Em um primeiro momento será analisado o *speedup* do algoritmo paralelo executado com diferentes números de *threads*, esta análise possibilita verificar o quanto a versão paralela acelera o processamento, além de permitir extrair dados de eficiência, garantindo a avaliação da utilização de recursos de processamento. Em seguida, para comparar o desempenho do PSAPSO em relação ao SAPSO, serão analisados o número de iterações e tempo de execução pois estes permitiram determinar o quão rápido os algoritmos conseguem encontrar a melhor solução, por fim a acurácia das soluções encontradas serão comparadas, com o objetivo de demonstrar a capacidade dos algoritmos encontrarem ótimos globais.

1.7 Estrutura do trabalho

Os demais capítulos deste trabalho estão organizados da seguinte maneira. No Capítulo 2 será apresentado o embasamento teórico que inspirou o desenvolvimento desta pesquisa, ressaltando principais algoritmos e técnicas presentes na literatura. O Capítulo 3 apresentará a metodologia utilizada no desenvolvimento deste estudo, detalhando as etapas necessárias para a implementação e testes dos algoritmos. No Capítulo 4, os resultados obtidos nos testes são apresentados e avaliados, demonstrando as vantagens e desvantagens do algoritmo. Finalmente, o Capítulo 5 conclui a pesquisa, levantando pontos de interesse e propondo ideias para trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica necessária para entender sobre PSOs e paralelização de algoritmos bioinspirados que foram importantes para a concepção da versão paralela do SAPSO. A princípio, será introduzido brevemente o algoritmo PSO canônico, algumas variantes do PSO e o SAPSO. Em seguida, serão avaliados os modelos de comunicação em arquiteturas paralelas aplicadas ao PSO, a metodologia de paralelização utilizada para segmentar de forma lógica o processo de paralelização e modelos de avaliação de processos concorrentes aplicados a esse estudo.

2.1 PSO canônico

O PSO foi concebido a partir do estudo de simulações de modelos sociais da natureza. Tomando como influência trabalhos publicados em áreas como computação gráfica e psicologia social, os autores Kennedy e Eberhart (KENNEDY; EBERHART, 1995) observaram o potencial da utilização de tais modelos para resolução de problemas de otimização.

Nesse método, é utilizado um conjunto de soluções possíveis, que são dispersas no espaço de busca de um problema, o qual é modelado como uma função matemática. Tais soluções trocam informações entre si, a fim de determinar a melhor solução para a função. Esse conjunto é denominado enxame e seus integrantes são chamados de partículas. As partículas podem ser interpretadas como uma estrutura de dados composta por três aspectos principais: o vetor n-dimensional velocidade $\vec{v}_i^t$, que determina a tendência de movimento de uma determinada partícula; o vetor posição $\vec{x}_i^t$, que especifica a posição atual da partícula no espaço de busca; e o *fitness*, que representa o quão apta é a solução em relação ao problema.

O algoritmo é executado de modo iterativo, onde, a cada iteração t, os vetores de velocidade e posição são atualizados. O processo de troca de informações ocorre durante o cálculo da velocidade, no qual uma partícula i utiliza a informação da posição $\vec{g}^t$ de maior *fitness*, encontrada pela melhor partícula G. A Equação 2.1 define o modelo matemático que rege a forma com que a velocidade é calculada, a qual conta com três componentes: a primeira corresponde ao vetor velocidade na iteração atual; a segunda é denominada componente cognitiva, onde é feita uma soma vetorial entre a posição atual e a melhor posição encontrada pela partícula; e a terceira componente representa o comportamento social da partícula, que executa uma soma vetorial entre a melhor solução encontrada pelo enxame e a posição atual da partícula. As constantes c_1 e c_2 correspondem, respectivamente, aos coeficientes cognitivo e social, φ_1 e φ_2 são variáveis aleatórias uniformes, com valores

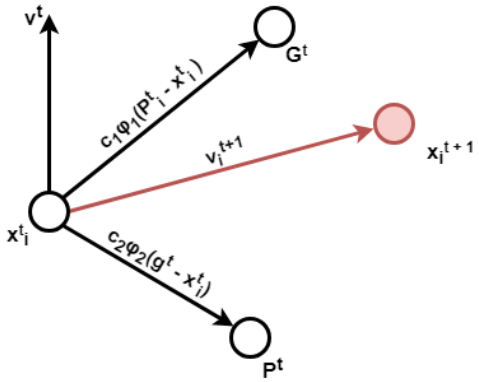
no intervalo [0,1], que determinam a influência dos componentes e $\vec{P}_i^t$ representa o melhor resultado encontrado pela partícula.

$$\vec{v}_i^{t+1} = \vec{v}_i^t + c_1\varphi_1(\vec{P}_i^t - \vec{x}_i^t) + c_2\varphi_2(\vec{g}_i^t - \vec{x}_i^t) \tag{2.1}$$

A cada iteração do algoritmo, todas as partículas atualizam seus vetores de posição para a próxima iteração $t + 1$, este processo é executado a partir de uma simples soma vetorial, como demonstrado na Equação 2.2. A Figura 1 exemplifica a atualização da velocidade e da posição uma partícula a partir das três componentes previamente descritas.

$$\vec{x}_i^{t+1} = \vec{x}_i^t + \vec{v}_i^{t+1} \tag{2.2}$$

Figura 1 – Exemplificação da atualização dos vetores de posição e velocidade.



Fonte: Elaborado pelo autor.

2.2 Semi-Autonomous Particle Swarm Optimization (SAPSO)

Visando a melhoria do processo de otimização, facilitando a fuga de mínimos locais e aprimorando os processos de *exploration* e *exploitation*, o SAPSO busca integrar diferentes operadores, como diversidade, cálculo do gradiente e um modelo de atração e repulsão das partículas. O funcionamento do SAPSO é centrado na avaliação de métricas que possam ser usadas como critérios de controle de cada partícula e do enxame como um todo, onde cada partícula possui certo nível de autonomia para determinar o modelo que deve utilizar para navegar o espaço de busca. Porém, ao mesmo tempo que cada partícula possui autonomia para escolher o modo de navegação, o comportamento do enxame é controlado por uma medida compartilhada entre as partículas, o que confere a característica de semi-autonomia ao algoritmo (SANTOS et al., 2018).

2.2.1 Cálculo do gradiente

Devido ao modo com que as partículas exploram o espaço de busca no algoritmo original de PSO, é recorrente o desperdício de poder computacional para avaliar posições repetidas do espaço de busca, devido a sua característica estocástica e dependência do melhor global. Portanto, como forma de orientar as partículas aos pontos de mínimo de modo determinístico, e assim reduzir os custos computacionais, é interessante a integração do cálculo do gradiente à função da velocidade do PSO, uma vez que o gradiente resulta em um vetor direcionado ao ponto de mínimo mais próximo à posição avaliada (NOEL, 2012). No contexto do SAPSO, a componente cognitiva, presente no PSO canônico, é substituída por uma componente que utiliza o vetor resultante do gradiente para orientar as partículas.

2.2.2 Gradiente e o melhor global

Durante a execução do algoritmo, as partículas podem assumir dois modos de investigação, em que a função de atualização da velocidade pode utilizar informações do melhor indivíduo do enxame ou usar o cálculo do gradiente. A escolha do método é dependente da avaliação da estagnação de uma partícula.

Por padrão, as partículas são iniciadas em posições aleatórias e utilizam o valor do gradiente, de forma que possam avaliar os mínimos locais mais próximos de suas posições originais. A cada iteração, é verificada a variação do valor do *fitness* e, caso uma partícula não demonstre melhoria significativa da solução ao longo de um determinado número de gerações, seu estado de navegação é alterado, passando a tomar como referência a posição do melhor indivíduo do enxame. Esse processo possibilita que a partícula escape de ótimos locais, uma vez que a influência do gradiente no cálculo da velocidade não permitirá que a partícula saia do vale explorado. A partícula retorna à exploração em função do gradiente ao se aproximar de uma distância que seja inferior a um parâmetro que delimita a distância mínima entre a partícula e o melhor global.

2.2.3 Atração e repulsão para controle da diversidade

Outro fator importante é a avaliação da diversidade do enxame, que representa a pluralidade de soluções em um mesmo conjunto de partículas. Os resultados da diversidade são utilizados como parâmetro para determinar a dispersão do enxame sobre o espaço de busca. Valores crescentes de diversidade significam o distanciamento entre as partículas, consequentemente, a exploração de posições distintas do espaço de busca, enquanto valores decrescentes da diversidade demonstram aproximação das partículas e inclinação para a convergência em um ponto de mínimo (VESTERSTRØM; RIGET, 2002).

O algoritmo é iniciado utilizando o processo de atração, onde cada partícula utiliza

sua própria informação do gradiente ou do melhor global, até o momento em que a diversidade do algoritmo chega a valores inferiores a um determinado limiar. Desse modo, o algoritmo verifica a diversidade e a utiliza para alternar entre os estados de atração e repulsão. Quando o valor da diversidade está abaixo de um limite mínimo predeterminado, o enxame entra em estado de repulsão, invertendo o sentido da influência do gradiente ou do melhor global. Por outro lado, quando o processo de repulsão ocasiona o crescimento da diversidade acima de um determinado limite máximo, as partículas retornam ao processo de atração.

2.2.4 Atualização da velocidade

Com os conceitos fundamentais introduzidos, é importante avaliar como estes são modelados no algoritmo, que são apresentados como alterações aplicadas à função de cálculo da velocidade. A Equação 2.3 apresenta o modelo de atualização da velocidade no SAPSO. Pode-se perceber que a primeira componente da função utiliza coeficiente de inércia dinâmico, denotado por w^t, para variar a influência do vetor velocidade ao longo do tempo.

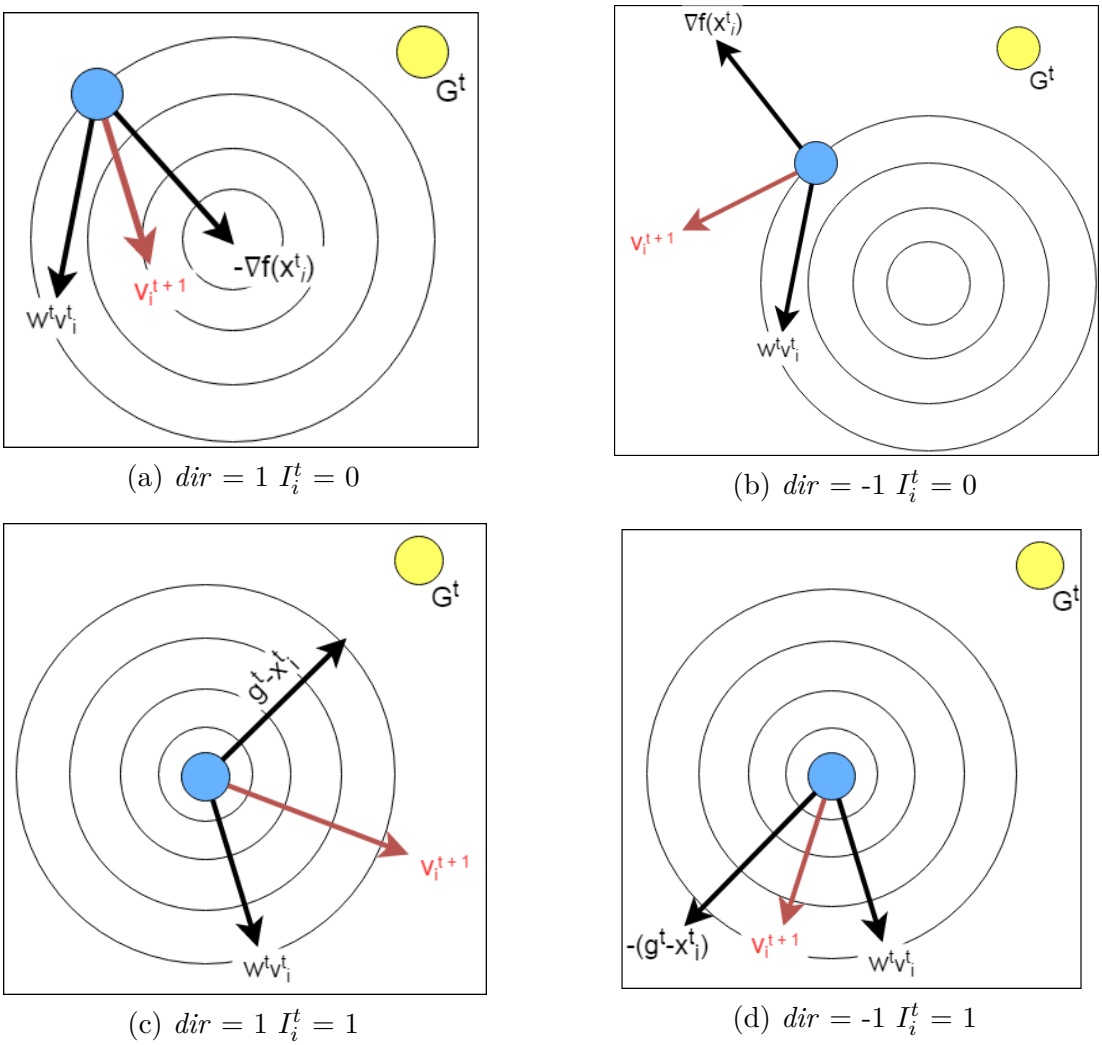
Os fatores que caracterizam o comportamento semi-autônomo do algoritmo estão contidos na segunda componente, que pode ser interpretada como a comunicação da partícula com o ambiente. Nesse contexto, as variáveis I_i^t e *dir* definem respectivamente o controle individual de cada partícula e o processo de atração e repulsão. A variável I_i^t é binária e privada para uma partícula i, que é responsável por determinar a informação que a partícula usará para navegar pelo espaço de busca. Percebe-se na Equação 2.3 que os valores dessa variável ativam as componentes social e do gradiente. A variável *dir*, no entanto, é compartilhada entre todas as partículas e pode assumir os valores 1 para representar a atração entre as partículas e o valor -1 para a repulsão, já que esta influenciará diretamente na direção do vetor resultante da avaliação do ambiente.

$$\vec{v}_i^{t+1} = w^t \vec{v}_i^t + dir * [I_i^t c_1 \varphi_1 (\vec{g}^t - \vec{x}_i^t) + (I_i^t - 1) c_2 \varphi_2 \nabla f(\vec{x}_i^t)] \tag{2.3}$$

A utilização desses operadores possibilita balancear entre *exploration* e *exploitation*. A Figura 2 ilustra o vetor velocidade resultante (seta vermelha) em todos os estados possíveis que podem ser assumidos pelas partículas no SAPSO. A Figura 2a representa o caso em que $dir = 1$ e $I_i^t = 0$, onde a velocidade é calculada a partir da soma vetorial da componente da velocidade e a componente do valor negativo do gradiente da função, que é direcionado ao vale mais próximo. A Figura 2b apresenta o caso em que dir = -1 e $I_i^t = 0$, onde a redução da diversidade do enxame iniciou o processo de repulsão, portanto, o cálculo da velocidade utilizará o valor inverso do gradiente, possibilitando que a partícula avalie novas posições do espaço de busca, contribuindo para o processo de *exploration*. A

configuração $dir = 1$ e $I_i^t = 1$ é demonstrada na Figura 2c, onde a partícula se encontra estagnada em um mínimo local por não detectar melhorias consideráveis em seu próprio *fitness*, assim, esta desconsidera a informação do gradiente e passa a utilizar os dados do melhor global, em um processo que possibilita que a partícula fuja de mínimos locais. Por fim, a Figura 2d ilustra o cenário em que dir = -1 e $I_i^t = 1$, onde a direção oposta ao melhor global é considerada; semelhante ao caso da Figura 2b, esse caso possibilita a fuga do mínimo local e auxilia o processo de *exploration*.

Figura 2 – Possíveis estados das partículas no SAPSO.



(a) $dir = 1$ $I_i^t = 0$

(b) dir = -1 $I_i^t = 0$

(c) $dir = 1$ $I_i^t = 1$

(d) dir = -1 $I_i^t = 1$

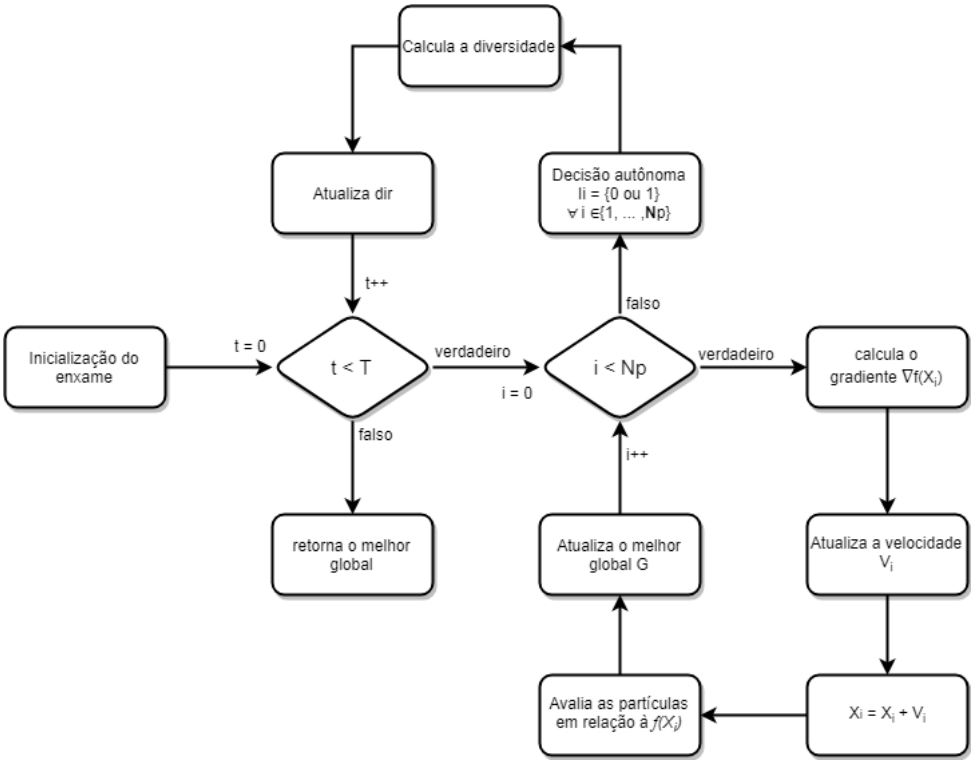
Fonte: Elaborado pelo autor.

2.2.5 Fluxo de execução

Um fluxograma com a organização da execução é apresentado na Figura 3. O algoritmo apresenta uma estrutura semelhante à de demais variações do PSO, que conta com o aninhamento de dois *loops*. O loop mais interno opera sobre um vetor de tamanho Np que armazena o enxame de partículas, o qual executa os cálculos da velocidade, posição

e do gradiente para cada partícula i. Nesse loop, também é feita a eleição do melhor indivíduo do enxame. No mais externo, comumente chamado de loop principal, ocorre a execução do loop de avaliação do enxame, conforme descrito anteriormente. Além disso, nesse ponto são processadas as operações aplicadas a todo o enxame, como a decisão autônoma, a decisão coletiva e o cálculo da diversidade.

Figura 3 – Fluxograma do SAPSO.



Fonte: (SANTOS et al., 2018)

2.3 Metodologia PCAM

Computação paralela é um projeto de algoritmo que busca melhorar o desempenho de processos quanto ao tempo de execução e a possibilidade de manipular grandes quantidades de dados simultaneamente. No entanto, isso exige uma grande atenção ao modo com que os algoritmos são modelados e implementados, uma vez que a má utilização de recursos paralelos pode comprometer as vantagens do paralelismo, já que esses algoritmos são suscetíveis a problemas como condição de corrida ou *deadlock*.

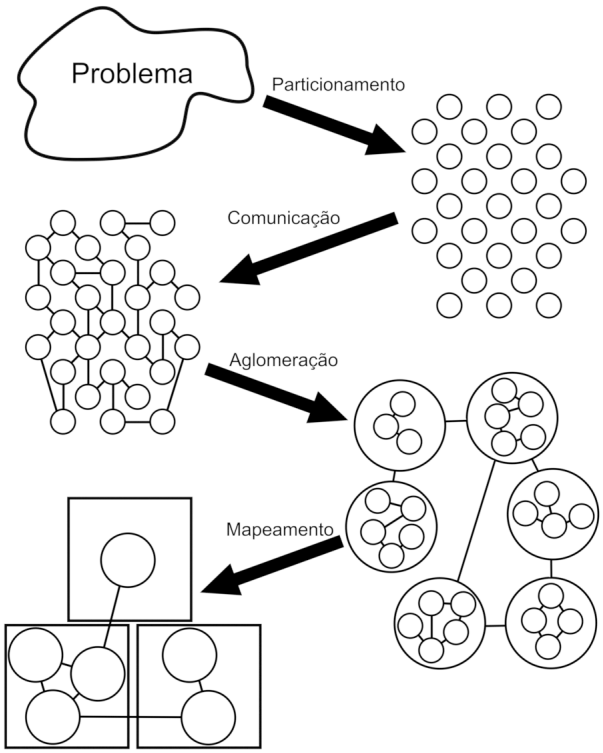
Tendo em mente a complexidade de implementar processos paralelos, Ian Foster (FOSTER, 1995) apresentou uma metodologia que tem como finalidade estruturar em etapas o processos de paralelização e ajudar a encontrar gargalos e problemas na implementação. O modelo é identificado pelo acrônimo PCAM, em que cada letra é associada

com uma etapa do processo, sendo essas particionamento, comunicação, aglomeração e mapeamento.

O fluxo do modelo é apresentado de modo geral na Figura 4. O processo é iniciado a partir da especificação de um problema com dados conhecidos. Em seguida, é feita a etapa de particionamento, em que as tarefas necessárias para solucionar o problema são divididas em tarefas menores e avaliadas em busca de oportunidades de paralelização, seja por divisão de dados ou tarefas. Nesta etapa não é levado em consideração o ambiente em que o algoritmo será implementado. Com o particionamento definido, dá-se início ao processo de comunicação, onde estratégias de troca de informação devem ser avaliadas com relação à adequação ao problema abordado. Alguns métodos presentes na literatura serão descritos mais detalhadamente na seção seguinte.

O processo de aglomeração, utiliza as tarefas divididas na etapa de particionamento e o modelo escolhido na comunicação e avalia os custos de implementação em um sistema computacional real, levando em conta parâmetros como o tamanho das tarefas e a frequência com que as tarefas se comunicam. Caso o modelo se mostre muito custoso, as tarefas são combinadas em tarefas maiores. Finalmente, é iniciada a etapa de mapeamento, onde as tarefas são alocadas aos processadores, buscando agrupar tarefas que se comunicam com frequência em um mesmo processador, enquanto tarefas que podem ser executadas concorrentemente devem ser atribuídas a processadores diferentes.

Figura 4 – Diagrama do modelo PCAM.



Fonte: Adaptado da página Designing and Building Parallel Programs[1]. - Autor: Ian Foster

2.4 Modelos de comunicação paralela

Ferramentas de desenvolvimento paralelo comumente disponibilizam modos de comunicação entre processos paralelos. Seja por meio de passagem de mensagem ou compartilhamento de memória, a implementação de programas paralelos requer algum meio de comunicação entre processos. Para algoritmos de busca paralela, esse é um fator fundamental para sinalizar resultados encontrados ou para a cooperação entre processos. Em problemas de busca em conjuntos que comportam elementos independentes uns dos outros, é necessário que os processos se comuniquem para sinalizar que a solução foi encontrada. Já para casos de elementos interdependentes, a utilização de um modelo de comunicação contribui para a formação de uma rede de cooperação que objetiva localizar a solução.

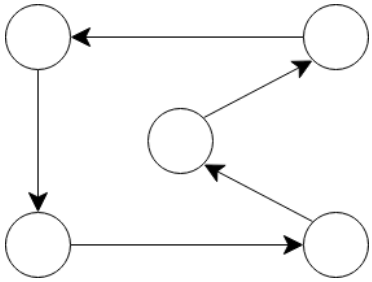
O PSO pode ser visto como propício à paralelização. Dependendo da abordagem, é possível associar uma unidade de processamento para cada partícula, ou utilizar múltiplos enxames que são executados em paralelo. Independente do contexto, os resultados do algoritmo têm forte dependência do modo com que as partículas se comunicam, portanto, é necessário escolher um modelo de comunicação apropriado para que a troca de informações seja feita de modo eficiente. A seguir são apresentados alguns modelos de paralelização aplicados à enxame de partículas (MUSSI; DAOLIO; CAGNONI, 2011; LALWANI et al., 2019).

2.4.1 Modelo de ilhas

A comunicação por meio de ilhas é caracterizada pela existência de múltiplos enxames que exploram o espaço de busca e avaliam suas partículas de forma independente. No entanto, para garantir a diversidade das soluções, os enxames trocam suas melhores partículas entre si. O modo com que a troca ocorre é determinado por uma topologia que deve ser definida previamente, representado, na Figura 5, como um grafo direcionado onde os vértices representam os enxames e as arestas as comunicações. O processo de escolha da topologia pode ter grande impacto no tempo de execução do algoritmo, devido ao custo computacional para executar a troca de informação entre enxames (ABADLIA; SMAIRI; GHEDIRA, 2017).

[1] Disponível em:<https://www.mcs.anl.gov/ itf/dbpp/text/node15.html> Acesso em Jul. 2019

Figura 5 – Modelo de comunicação por ilhas.

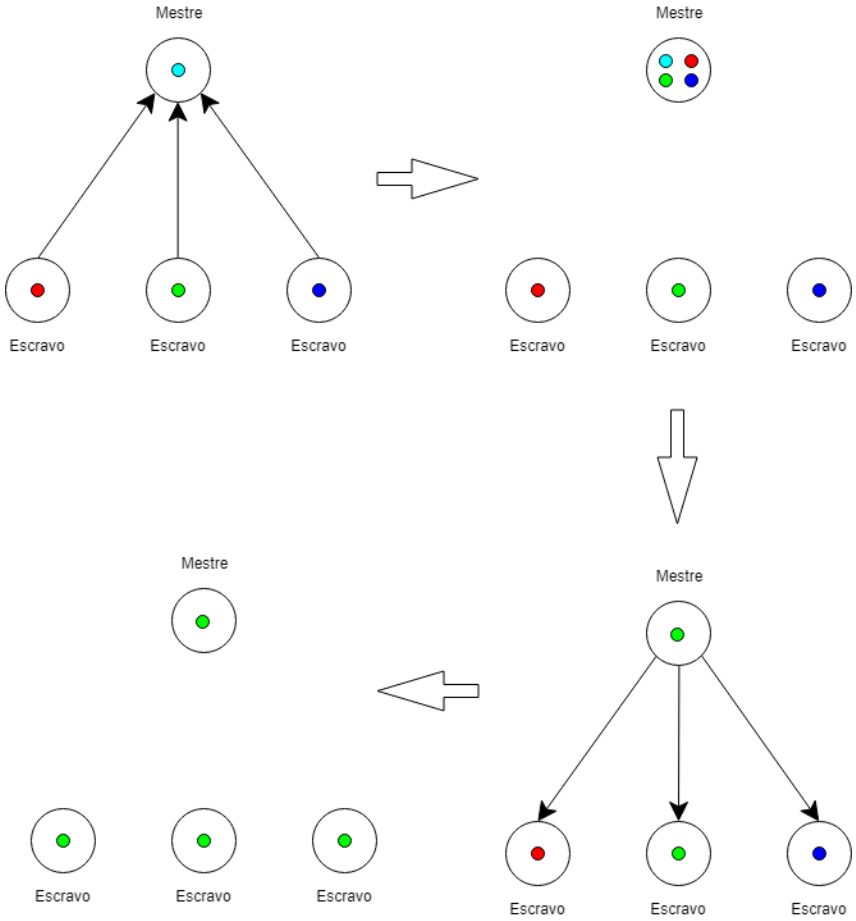


Fonte: Elaborado pelo autor.

2.4.2 Modelo mestre-escravo

O modelo mestre-escravo, apresentado na Figura 6, trata da divisão do subconjunto de enxames paralelos em dois grupos: um subconjunto unitário contendo o enxame classificado como mestre e um subconjunto contendo os demais enxames que passarão a ser chamados de escravos. A comunicação é feita por intermédio do mestre, onde este recebe as melhores partículas de todos os demais enxames e as armazena em um conjunto juntamente a sua própria melhor partícula. Em seguida é selecionada aquela com o melhor *fitness*, que será enviada aos escravos e adicionada a todos os enxames em uma posição aleatória (GüLCü; KODAZ, 2015).

Figura 6 – Método de troca de informação no modelo mestre-escravo. Os enxames escravos compartilham suas melhores partículas com o mestre. O mestre avalia as partículas recebidas distribui a partícula de maior fitness para todos os enxames.

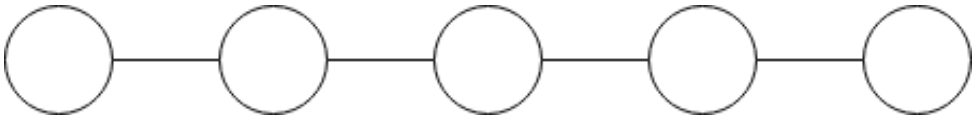


Fonte: Elaborado pelo autor.

2.4.3 Modelo celular

O modelo celular é o mais simples entre os modelos apresentados. Nesse modelo os enxames são dispostos em um vetor bidimensional, limitando a comunicação apenas entre os enxames em posições adjacentes. Isso pode afetar a eficiência da transferência de dados, devido a utilização de uma única via de comunicação (Figura 7) (ABDULLAH; SALEH; SAIF, 2018)

Figura 7 – Modelo de comunicação celular.



Fonte: Elaborado pelo autor.

2.5 Modelos de avaliação de aplicações paralelas

Uma vez que a computação paralela se propõe a melhorar processos, são necessários métodos para mensurar os ganhos de versões concorrentes em relação à suas contrapartes executadas em um único núcleo. Portanto, nesta seção serão apresentadas métricas e tópicos de interesse para análise de algoritmos paralelos.

2.5.1 Speedup

Speedup é definido como o ganho em velocidade de uma aplicação, em relação à sua versão sequencial, quando executada em um ambiente paralelo. Em uma situação ideal, onde os processos podem ser completamente paralelizados e os custos computacionais para troca de informações e criação de *threads* são inexistentes, o ganho de velocidade é dado por n, onde n representa o número de núcleos utilizados no processo. No entanto, sabe-se que estas aplicações não são imunes a custos de comunicação ou manipulação de *threads*, portanto o *speedup* real é calculado como a razão do tempo de execução sequencial Ts pelo tempo de execução paralelo Tp, apresentado na Equação 2.4.

$$Speedup = \frac{T_s}{T_p} \tag{2.4}$$

2.5.2 Eficiência

A eficiência é matematicamente definida como a razão do *speedup* em relação ao número N de processadores envolvidos no processo, e é apresentada na Equação 2.5. Esta determina a produtividade dos N núcleos em relação ao modo com que a carga de trabalho está dividida. Para casos em que a eficiência é igual a 1, cada núcleo está ocupado durante todo o processo de execução e as tarefas estão igualmente divididas.

$$Eficiência = \frac{Speedup}{N} \tag{2.5}$$

2.5.3 Lei de Amdahl

Em 1967 foi apresentado pelo cientista da computação Gene Amdahl (AMDAHL, 1967) a teoria que fundamentou o modelo matemático que passaria a ser conhecido como a lei de Amdahl, a qual modela o dito *speedup* teórico, que delimita o ganho máximo de velocidade obtido no processo de paralelização.

A lei de Amdahl leva em consideração uma tarefa dividida em duas partes: a parte serial, que caracteriza um aspecto específico da tarefa que não pode ser subdividido, e a porção paralela, correspondente às tarefas que serão efetivamente compartilhadas entre os múltiplos processadores, denotada pelo caractere α. Outro fator importante é que a lei

desconsidera o *overhead* de comunicação e particionamento, possibilitando assim dividir a porção paralela igualmente entre os N núcleos (BARLAS, 2015). É afirmado que o tempo de execução de um processo paralelo é limitado pelo produto do tempo de execução serial e a soma das partes serial e paralela do algoritmo, demonstrada na Equação 2.6.

$$T_p = T_s * [(1 - \alpha) + \frac{\alpha}{N}] \tag{2.6}$$

Substituindo o tempo de execução paralelo T_p da lei de Amdahl, na função de *speedup*, temos a Equação 2.7, que define o limite máximo atingido pelo *speedup* de um processo paralelo.

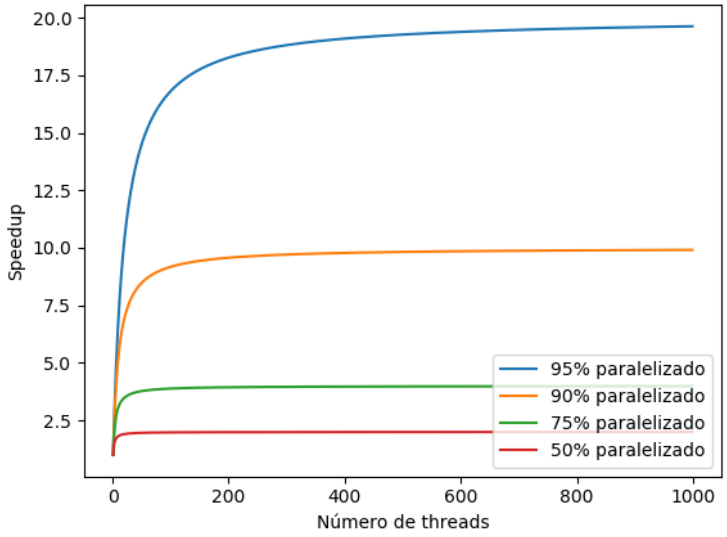
$$Speedup = \frac{1}{[(1 - \alpha) + \frac{\alpha}{N}]} \tag{2.7}$$

Uma vez que a lei desconsidera *overheads* adicionais inerentes ao processo de paralelização, é de interesse avaliar o comportamento da função em relação ao aumento do número N de núcleos de processamento. O *speedup* máximo será obtido quando $N \to \infty$, resultando na Equação 2.8, que demonstra que o tempo de execução da fração paralela será insignificante e o *speedup* será limitado pelo tempo de execução da fração sequencial do algoritmo.

$$lim_{N \to \infty} = \frac{1}{(1 - \alpha)} \tag{2.8}$$

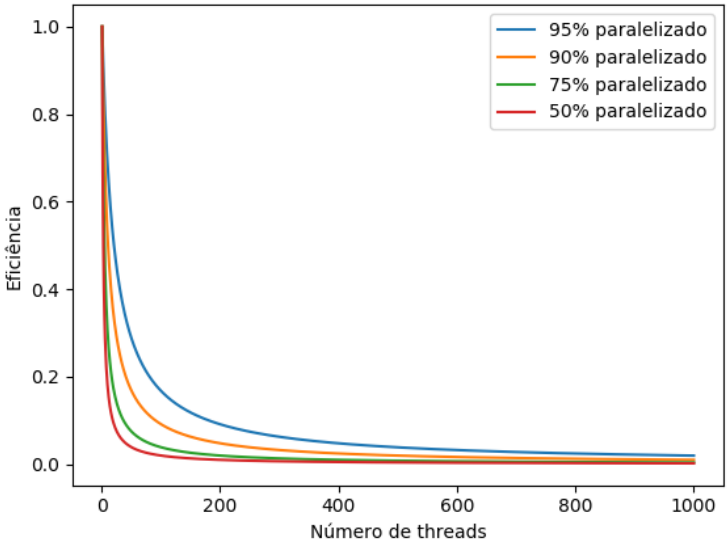
A Figura 8 apresenta a representação gráfica da lei de Amdahl. Nota-se que o *speedup* se torna constante a medida que o número de processadores N cresce o suficiente. Esse gráfico não reflete o comportamento real do *speedup*, uma vez que este está sujeito a custos computacionais para criação e coordenação de *threads*. A Figura 9 mostra o impacto negativo da lei de Amdahl sobre a eficiência. Verifica-se que o aumento excessivo do número de núcleos causará na redução da eficiência, ocasionado pela grande fragmentação de tarefas e o mal aproveitamento dos recursos de processamento.

Figura 8 – Lei de Amdahl aplicada ao *speedup*.



Fonte: Elaborado pelo autor.

Figura 9 – Lei de Amdahl aplicada à eficiência.

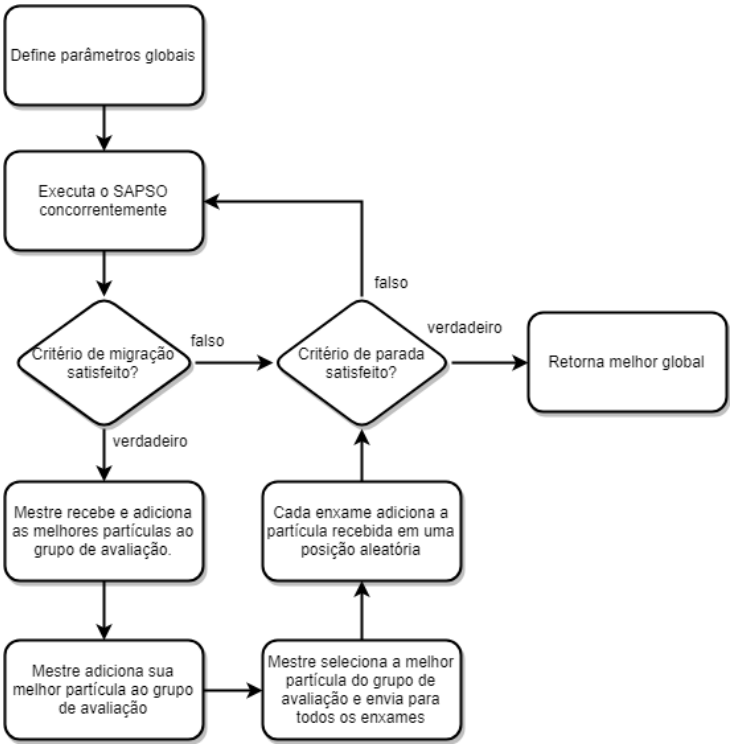


Fonte: Elaborado pelo autor.

3 Otimizador Paralelo de Enxame de Partículas Semi-Autônomas

Este capítulo apresenta a aplicação dos conceitos e processos apresentados no capítulo anterior ao SAPSO, com o objetivo de expandir as suas capacidades no que diz respeito ao tempo de processamento e qualidade das soluções. A Figura 10 ilustra o fluxo de execução iterativa do algoritmo desenvolvido. Nesse múltiplas instâncias do SAPSO são executadas concorrentemente, que utilizam um modelo mestre-escravo para difundir informações do espaço de busca.

Figura 10 – Diagrama de fluxo do PSAPSO.



Fonte: Adaptado de (GüLCü; KODAZ, 2015)

O ambiente de desenvolvimento é um conjunto de ferramentas computacionais que fundamentam todo o processo de criação de softwares. No entanto, esse espaço é geralmente orientado à implementação serial. Assim, para a criação de programas paralelos, é necessário utilizar recursos que habilitem a utilização e controle de múltiplos processos. A seguir é apresentada a configuração do ambiente utilizada no desenvolvimento do projeto, demonstrando suas vantagens e utilidades no processo de criação do PSAPSO.

3.1 Ambiente de desenvolvimento

O algoritmo original do SAPSO (SANTOS, 2018) foi desenvolvido utilizando o software para cálculos matemáticos MATLAB (MATHWORKS, 2019). No entanto, devido a se tratar de uma ferramenta proprietária, a dificuldade de acesso a licenças de uso se mostra uma barreira para o desenvolvimento e divulgação de trabalhos científicos. Como alternativa, optou-se por utilizar a linguagem C++, por ser uma linguagem aberta, e que é amplamente utilizada na indústria e academia (STANDARD C++ FOUNDATION, 2019).

Outro ponto importante de C++ é que, além do suporte nativo a paralelismo (oferecido a partir da revisão C++11), também possui compatibilidade com outras APIs, bibliotecas e frameworks para paralelização. Para a implementação do SAPSO, foi escolhida a API OpenMP (OPENMP, 2019), em função da facilidade de uso e extensa documentação disponível na Internet.

Para que fosse possível utilizar a versão mais recente do OpenMP, foi necessário a utilização do sistema operacional Linux, uma vez que o suporte ao sistema operacional Windows foi descontinuado em 2002. A implementação deste projeto foi feita na distribuição Linux Kubuntu 19.04 (KUBUNTU DEVS, 2019).

A compilação dos códigos escritos em C++ foi feita utilizando o compilador GCC (FREE SOFTWARE FOUNDATION, 2019a), em função de sua facilidade de acesso. Para depuração, foi utilizado o depurador GDB (FREE SOFTWARE FOUNDATION, 2019b), o qual não foi criado tendo em mente aplicações paralelas, então apresenta limitações para avaliar esses tipos de aplicações. Ao tentar depurar códigos paralelos, o GDB intercala o processo de depuração entre as múltiplas *threads* executadas, ocasionando confusões e perda de eficiência no rastreio de possíveis bugs durante o processo.

Por fim, para unificar todas as ferramentas, foi utilizada a IDE Qt Creator (THE QT COMPANY, 2019). Esta possibilitou criar um ambiente de desenvolvimento integrado, agilizando o processo de implementação e testes.

3.2 Aplicação da metodologia PCAM

A metodologia PCAM foi utilizada para padronizar o procedimento de paralelização do SAPSO e, consequentemente, facilitar sua documentação. Seguindo a segmentação proposta pela metodologia, em que inicialmente deve ser identificado o problema abordado, foi considerado o SAPSO como o problema principal a ser paralelizado. Uma vez com o problema identificado, dá-se início às demais fases. Abaixo é descrito como cada etapa é inserida no contexto da implementação do PSAPSO.

3.2.1 Particionamento

A partir da avaliação da literatura, pôde-se perceber que existem duas táticas de particionamento mais comuns aplicadas ao PSO, que são particionamento de dados e de espaço (CICIRELLI et al., 2015). No particionamento de espaço, o espaço de busca é particionado e atribuído aos núcleos de processamento em que cada núcleo é responsável pelas partículas que residem na sua parcela do espaço. Esse modelo não apresenta melhorias da solução, mas pode ter grande impacto na redução do tempo de processamento. Por esse motivo, a paralelização do espaço foi desconsiderada, visto que o objetivo desta pesquisa é melhorar a capacidade de busca em conjunto com o tempo de execução.

Em contrapartida, o particionamento de dados, em que tarefas como atualização da velocidade ou o próprio enxame podem ser distribuídos entre múltiplos núcleos, torna possível melhorar do tempo de execução. Para o caso de múltiplos enxames se torna possível a utilização de uma estrutura de transferência de informação, uma vez que cada núcleo pode compartilhar informações de seu enxame com os demais. Devido a esses fatores, optou-se por subdividir as tarefas do SAPSO em função dos dados processados.

Foram avaliados três métodos de subdivisão de tarefas relacionados ao particionamento de dados. Primeiramente, a utilização de um único enxame com a divisão das partículas em cada núcleo, onde esses seriam encarregados de operar sobre uma ou mais partículas. No entanto este é um modelo mais apropriado para execução em GPU, em função ao seu grande número de núcleos (DALI; BOUAMAMA, 2018). Essa opção foi desconsiderada, devido neste trabalho a implementação do algoritmo ser a partir de uma plataforma baseada em CPUs. Essa abordagem apresenta uma grande desvantagem devido a único núcleo representar uma grande quantidade de recursos, e a associação de um núcleo a um conjunto limitado de partículas pode resultar no subaproveitamento das capacidades do processador.

A segunda opção é a paralelização dos operadores do SAPSO. Nessa abordagem, o algoritmo é parcialmente serial e é iniciada a execução paralela sempre que for necessária a utilização de *loops* para iterar sobre o enxame. Essa opção também foi desconsiderada, por estar apenas relacionada a melhoria do tempo de execução.

Finalmente, a opção escolhida foi a utilização de múltiplos enxames, em que cada núcleo é responsável por um enxame. Essa opção pode ser vista como múltiplas execuções simultâneas do algoritmo, onde o número de execuções é determinado pelo número de *threads* utilizadas. Esse modelo apresenta grandes vantagens por possibilitar a cooperação de múltiplos enxames com conhecimentos distintos de um mesmo espaço de busca, a partir da utilização de modelos de compartilhamento de informações. A Figura 11 apresenta o trecho do código utilizado para definir esta estratégia. Na linha 131 é utilizada diretiva responsável por dividir o processamento entre múltiplas *threads*.

Figura 11 – Código para implementação da estratégio de múltiplos enxames.

```
131 #pragma omp parallel num_threads(threadNum)
132                 {
133                     int dir = 1;
134                     int globalBest = 0;
135                     double diversity = 0;
136                     std::vector<int> particleDecision(NPART, 0); // Decision of every particle.
137                     std::vector<int> stagnationCounter(NPART, 0); // Saves the number of stagnant generations.
138                     SAPSOServices services(functionName, seed[omp_get_thread_num()]);
139                     std::vector<Particle> swarm;
140 
141                     //Swarm and service initialization.
142                     for (int i = 0; i < NPART; i++)
143                     {
144                         Particle particle;
145 
146                         particle.setPosition(services.generateRandomSet(DIM, RANGE));
147                         particle.setVelocity(services.generateRandomSet(DIM, VMAXSet));
148                         particle.setFitness(services.evaluateParticle(particle.getPosition()));
149                         particle.setPersonalBestPosition(particle.getPosition());
150                         particle.setPersonalBestFitness(particle.getFitness());
151                         particle.setOldFitness(particle.getFitness());
152 
153                         swarm.push_back(particle);
154                     }
155 
156                     globalBest = services.getGlobalBest(swarm);
157 
158                     //Main loop.
159                     for (int i = 0; i < MAXITER; i++)
160                     {
```

Fonte: Elaborado pelo autor.

3.2.2 Comunicação

Para integrar os múltiplos enxames especificados na etapa de particionamento, foi adotado o modelo proposto por Gülcü e Kodaz (GüLCü; KODAZ, 2015), que se trata de uma estrutura mestre-escravo. Nessa estrutura os enxames a partir da transferência de suas melhores partículas para um processo de seleção efetuado no enxame mestre. A utilização da API OpenMP permitiu implementar esse processo a partir da criação de posições de memória compartilhada entre as *threads* utilizadas, o código utilizado na implementação é demonstrado na Figura 12. Nas linhas 195 e 208 são utilizadas diretivas que forçam que as *threads* interrompam suas execuções até que todas estejam neste mesmo ponto.

Figura 12 – Código para implementação do processo de comunicação entre enxames.

```
192                     if( (i + 1) % migrationInterval == 0 || i == MAXITER - 1 )
193                     {
194                         sharedParticles[omp_get_thread_num()] = &swarm[globalBest];
195 #pragma omp barrier
196 #pragma omp master
197                         {
198                             int best = 0;
199
200                             for(int n = 1; n < threadNum; n++)
201                             {
202                                 if(sharedParticles[best]->getFitness() > sharedParticles[n]->getFitness())
203                                     best = n;
204                             }
205
206                             swarmBest = *sharedParticles[best];
207                         }
208 #pragma omp barrier
209
210                         globalBest = randomizer.RandInt(NPART);
211                         swarm[globalBest] = swarmBest;
212                     }
213
214                     if (swarm[globalBest].getPersonalBestFitness() < STOPC)
215                     {
216                         swarmBest = swarm[globalBest];
217                         #pragma omp cancel parallel
218                     }
```

Fonte: Elaborado pelo autor.

O funcionamento do modelo gira em torno da eleição de um núcleo como mestre, o qual ficará responsável por receber e avaliar os melhores globais de todos os enxames incluídos na execução do PSAPSO. A princípio, o algoritmo é executado de modo sequencial, para que sejam instanciadas e inicializadas as variáveis globais. Ao chegar no *loop* principal do algoritmo, a execução paralela é inicializada, e, a partir desse ponto, os núcleos do processador são ativados. Em seguida, cada núcleo inicializa seu próprio enxame e executa concorrentemente o SAPSO.

Devido à necessidade de coordenar as threads para a troca de informações, foram utilizados métodos de sincronização, que proporcionaram coordenar as *threads* de modo que a execução só será continuada quando todas as *threads* estiverem em um mesmo ponto do processo. Esse efeito caracteriza o algoritmo como síncrono. Tal método foi escolhido para o trabalho desenvolvido, particularmente por apresentar uma implementação simples.

A troca de informações é determinada pelo período de migração, que é definido por um número fixo de iterações. Esse é um fator que deve ser avaliado cautelosamente, uma vez que períodos de comunicação curtos podem ocasionar no aumento da qualidade da solução mas aumentar o *overhead* de processamento devido a alta frequência de troca de informações. Enquanto que períodos muito longos podem ocasionar na deterioração da qualidade da solução, porém melhorar o tempo de processamento. A Figura 12 mostra na linha 192 a utilização de uma estrutura condicional para verificar a chegada ao critério de parada. Nessa mesma estrutura é verificado se a iteração atual é igual ao número máximo de iterações, esse critério é utilizado como forma de garantir a eleição da partícula que

será definida como melhor solução encontrada, quando o número máximo de iterações não for um valor divisível pelo intervalo de migração

3.2.3 Aglomeração

A etapa de aglomeração não apresenta impacto significativo no processo de paralelização, pois cada núcleo executa o SAPSO de modo concorrente, que em sua versão sequencial já apresenta bom desempenho e não apresenta grandes custos computacionais.

3.2.4 Mapeamento

O mapeamento determina a organização da distribuição de enxames entre as *threads*. Como o enxame de maior importância no mapeamento é o mestre, esse deve possuir um identificador que o diferencie dos demais. Dessa forma, o mapeamento foi limitado pela estrutura de organização de *threads* na API OpenMP, onde é utilizo, por padrão, o identificador zero para definir a *thread* mestre. Na Figura 12, na linha 196, é exemplificada a implementação do mapeamento, onde é utilizada uma diretiva para delimitar a execução da avaliação de partículas apenas na *thread* mestre.

3.3 Dados coletados

Para avaliar o desempenho do PSAPSO com relação ao tempo de execução e qualidade das soluções, é necessário o armazenamento e avaliação de dados oriundos do algoritmo. Para cada caso de teste avaliado, foram coletadas as médias do tempo de execução, do *fitness* e do número de iterações até a convergência.

O tempo de execução permite utilizar as equação de *speedup* e eficiência, apresentadas no capítulo anterior, para quantizar o ganho de velocidade em relação à versão sequencial e o aproveitamento de recursos paralelos. É importante ressaltar que, por se tratar de um algoritmo síncrono, o tempo de execução será determinado pela thread mais lenta. Por isso, espera-se que, para a situação em que a versão paralela utilize o mesmo número de partículas por enxame que a sua contraparte sequencial, os tempos de execução das duas sejam semelhantes. Logo, a avaliação do *speedup* foi feita levando em consideração que o produto do número total de partículas n pelo número de *threads* N, na versão paralela, deve ser igual ou maior que número de partículas na versão sequencial.

O cálculo da eficiência permitiu avaliar um comportamento incomum no uso de recursos dos núcleos. Para alguns casos de teste, a eficiência ultrapassou o valor de um, o que poderia ser interpretado como se os núcleos estivessem operando acima de suas capacidades máximas. No entanto, esse é um caso possível, uma vez que, em certas ocasiões, influências da arquitetura, como proximidade de instruções na memória ou bom uso da

memória cache, podem cooperar para uma redução no tempo de execução, o que resulta em um speedup acima do valor linear, ou seja, super-linear (RISTOV et al., 2016).

A análise das médias do fitness e da iteração são fundamentais para determinar a influência da utilização do paradigma paralelo na solução final. Para avaliá-los, foi utilizado o teste estatístico de Wilcoxon, que oferece meios de verificar a existência de diferenças significativas entre os resultados obtidos no SAPSO e no PSAPSO.

3.4 Funções de *benchmark*

Com o propósito de testar a capacidade de exploração do PSAPSO, esse foi submetido a testes com funções de *benchmark*, onde foram analisadas a acurácia das soluções e o número de iterações até a convergência. As funções utilizadas nesta etapa foram baseadas nas apresentadas por Santos (SANTOS et al., 2018), que, por sua vez, foram retiradas do trabalho de De Jong (JONG, 1975). A Tabela 1 apresenta os modelos matemáticos e o intervalo de busca utilizados.

As funções f_1 e f_6 são unimodais, convexas e sem mínimos locais, portanto, de fácil solução. A função f_6 representa a aplicação de um conjunto de escalares ao vetor n-dimensional presente na função f_1, característica essa que ressalta a semelhança entre as funções. Por se tratarem de cenários simples, essas funções podem ser utilizadas como patamar inicial para demonstrar a eficácia da utilização do gradiente na exploração.

A função f_2 é uma função não convexa, originalmente utilizada por De Jong como uma função unimodal para teste de algoritmos genéticos. No entanto, trabalhos mais recentes afirmam a existência de um segundo mínimo, em cenários com muitas dimensões, o que a classifica como multimodal (GüLCü; KODAZ, 2015; SHANG; QIU, 2006). Grande parte da superfície dessa função é plana, o que a viabiliza como um modo eficiente de testar a exploração em situações com pouca influência da informação do gradiente.

As funções f_3 e f_4 são variações da função f_1 com a adição de componentes trigonométricas, resultando no aparecimento de múltiplos mínimos locais que circundam o mínimo global. São funções com forte potencial para avaliar a capacidade de fuga de mínimos locais.

A função f_5 é caracterizada pela existência de mínimos locais distribuídos por toda a sua superfície. Esta apresenta apenas um mínimo global, localizado em sua origem. Devido à grande densidade de mínimos locais, essa função é de grande importância para avaliar a fuga de mínimos locais.

A função f_7 é bidimensional e não convexa. Esta foi usada para avaliar o comportamento do algoritmo em cenários com poucas dimensões.

Por fim, f_8 trata de uma função fortemente multimodal, que foi utilizada para

avaliar a capacidade do algoritmo de encontrar soluções em espaços de busca com mínimo global descentralizado. Apesar da função ser centralizada, o intervalo de busca foi escolhido de modo a garantir a característica de descentralização.

Tabela 1 – Funções de *benchmark*

Nome	Equação	Intervalo
Sphere (f_1)	$\sum_{i=1}^{n} x_i^2$	-100 $\leq x_i \leq$ 100
Rosenbrock (f_2)	$\sum_{i=1}^{n}[b(x_{i+1} - x_i^2)^2 + (a - x_i)^2]$	-5.12 $\leq x_i \leq$ 5.12
Rastrigin (f_3)	$10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$	-5.12 $\leq x_i \leq$ 5.12
Griewank (f_4)	$1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$	-100 $\leq x_i \leq$ 100
Ackley (f_5)	$-a.exp(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(cx_i))$ + a + exp(1)	-50 $\leq x_i \leq$ 50
Ellipsoid (f_6)	$\sum_{i=1}^{n} ix_i^2$	-5.12 $\leq x_i \leq$ 5.12
Schaffer N.2 (f_7)	$0.5 + \frac{sin^2(x^2-y^2)-0.5}{(1+0.001(x^2+y^2))^2}$	-100 $\leq x_i \leq$ 100
Alpine (f_8)	$\sum_{i=1}^{n} \lvert x_i sin(x_i) + 0.1x_i \rvert$	0 $\leq x_i \leq$ 20

Fonte: Adaptado de (SANTOS et al., 2018)

4 Simulação numérica e análise de resultados

Tanto o SAPSO quanto o PSAPSO foram submetidos a dois cenários de testes distintos para que fosse possível coletar os dados para análise, que contaram com todas as funções de *benchmark* apresentadas no capítulo anterior, diferindo apenas na configuração dos parâmetros de execução. O primeiro cenário objetivou avaliar a influência do paralelismo no tempo de execução do PSAPSO. O segundo cenário buscou confirmar os efeitos do modelo de comunicação mestre-escravo sobre a qualidade da solução e a capacidade de convergência do algoritmo paralelo. Como forma de apoio às conclusões encontradas no segundo cenário, os resultados da execução dos dois algoritmos foram aplicados ao teste estatístico de Wilcoxon.

Em todos os cenários estudados, os parâmetros comuns ao SAPSO e ao PSAPSO foram definidos a partir dos especificados em (SANTOS et al., 2018). Para os coeficientes social c_1 e o coeficiente do gradiente c_2 foram utilizados valores específicos para cada função, apresentados na Tabela 2. O contador de estagnação $cMax$ foi definido como 3 e os limiares de diversidade inferior d_{low} e superior $dhigh$ foram determinados como 10^{-6} e 0,25, respectivamente. Quanto ao valor do intervalo de migração inerente ao PSAPSO, foi fixado em 20, tomando como referência os resultados apresentados em (GüLCü; KODAZ, 2015). Cada função de teste foi executada 20 vezes para 10, 20 e 30 dimensões, com exceção de Schaffer N.2, devido a ser uma função bidimensional.

Tabela 2 – Coeficientes social e do gradiente.

Função	Coeficiente social (c1)	Coeficiente do gradiente (c2)
Sphere (f_1)	2	10^{-2}
Rosenbrock (f_2)	2	10^{-3}
Rastrigin (f_3)	3	10^{-3}
Griewank (f_4)	3	10^{-2}
Ackley (f_5)	4	10^{-1}
Ellipsoid (f_6)	2	10^{-2}
Schaffer N.2 (f_7)	2	10^{-2}
Alpine (f_8)	4	10^{-1}

Fonte: Elaborado pelo autor

Os testes foram executados utilizando um sistema com as seguintes configurações: sistema operacional Kubuntu 19.04, processador AMD Ryzen 7 1700 3.0 GHz, e 16GB de memória RAM.

4.1 Análise do tempo de execução

Para alcançar o objetivo atribuído ao primeiro cenário, foram analisados o *speedup* e a eficiência, com base na média do tempo de execução. O cálculo do *speedup* permitiu encontrar o número ótimo de *threads* para maximizar os ganhos de velocidade. O número de *threads* encontrado foi utilizado como referência para a execução do segundo cenário de testes. A configuração desse cenário se deu da seguinte forma: foram executadas 1000, com único critério de parada sendo o número máximo de iterações. Visando aumentar o custo de processamento e facilitar a análise da diferença de processamento entre as versões do algoritmo, o número absoluto de partículas entre todas as threads foi definido como 200. A distribuição de partículas p por *threads* em relação ao número de *threads* N foi determinado pela Equação 4.1. Para garantir que o valor absoluto será igual ou maior que 200, é utilizado o arredondamento para cima. Foram comparados os resultados das execuções do algoritmo utilizando de 1 a 20 *threads*.

$$p = \lceil 200/N \rceil \tag{4.1}$$

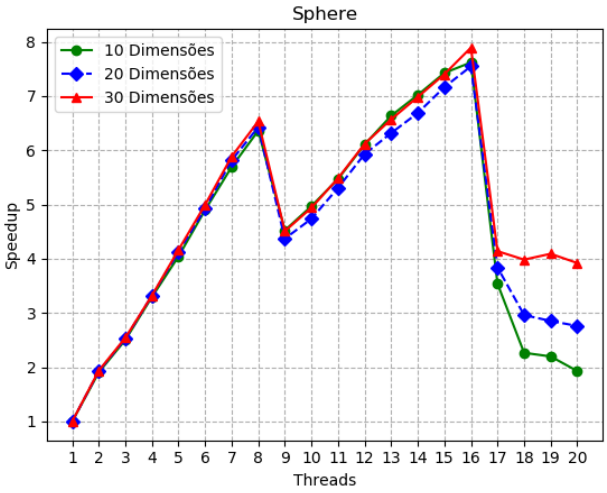
4.1.1 *Speedup*

A Figura 13 apresenta a análise do *speedup* no primeiro cenário. Em todos os casos de teste é aparente o comportamento sublinear do *speedup* para o número de *threads* entre 1 e 8, o que é esperado devido ao *overhead* ocasionado pela comunicação e criação de *threads*.

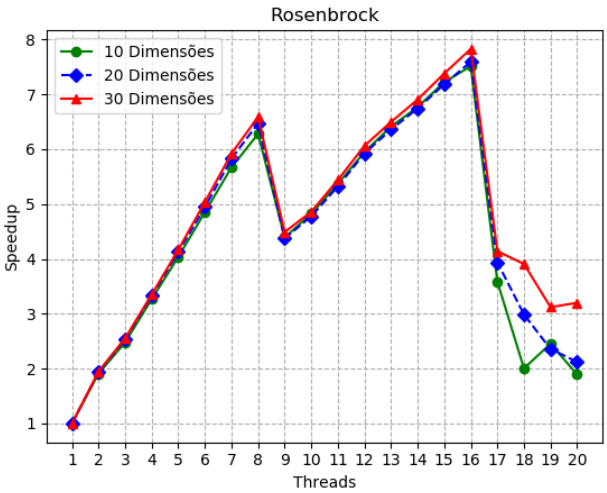
Para todas as funções, é evidenciada uma queda acentuada dos resultados obtidos com nove *threads*. Esse efeito é justificado pela arquitetura do processador com oito núcleos físicos e oito núcleos virtuais, que faz uso da tecnologia *Simultaneous Multithreading* (SMT) (VILANOVA; AMIT; ETSION, 2019), proprietária de processadores da AMD, que possibilita escalar a capacidade de processamento a partir da ativação dos núcleos virtuais. No entanto, esse processo ocasiona custos adicionais de tempo de processamento. O ganho de velocidade só compensa o custo de ativação dos núcleos virtuais a partir da utilização de 10 a 13 *threads*.

É possível perceber o pico do ganho de velocidade para a execução do algoritmo até 16 *threads*, após esse valor, os resultados demonstram perda significativa de velocidade. Com base nesses resultados, os testes do segundo cenário, apresentado na subseção 4.2, foram executados mantendo constante o número de *threads* em 16.

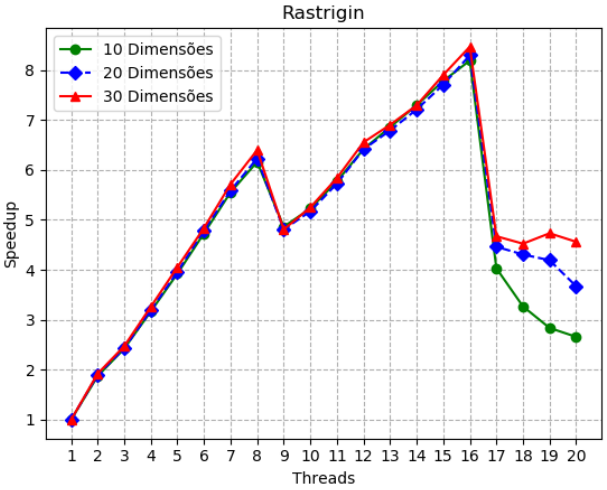
Figura 13 – Resultados de *speedup*.



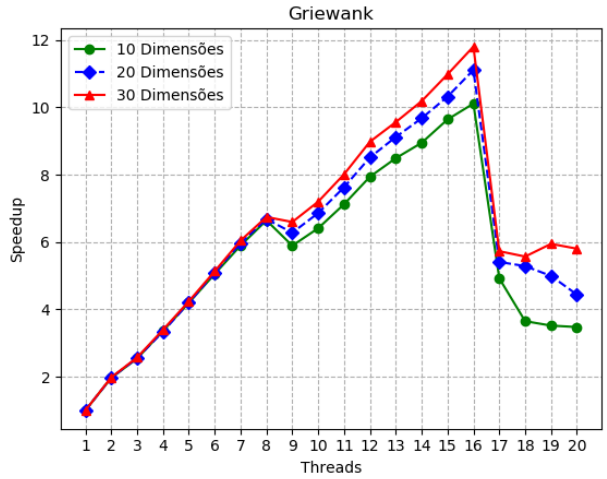
(a) f_1



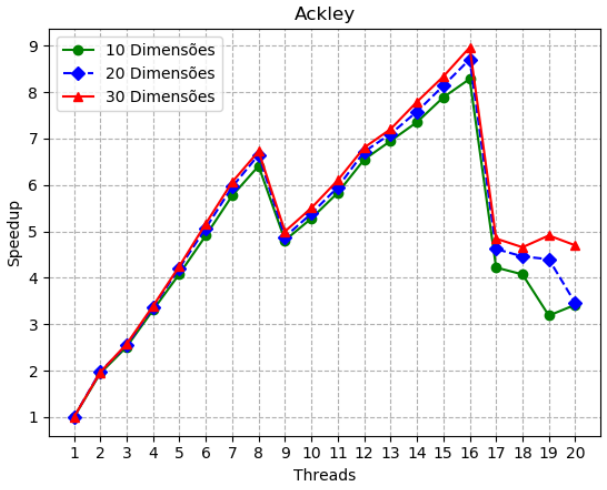
(b) f_2



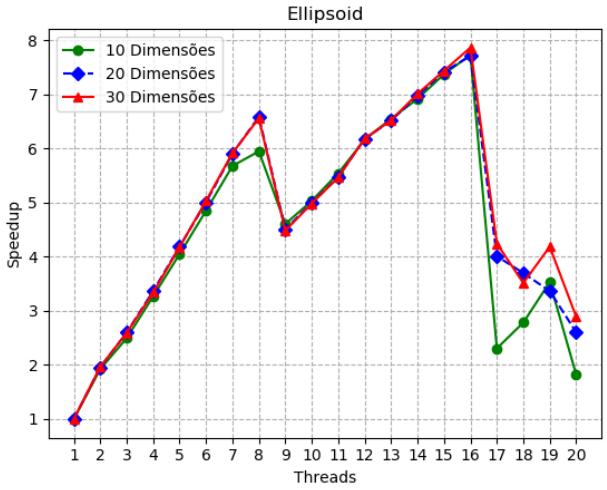
(c) f_3



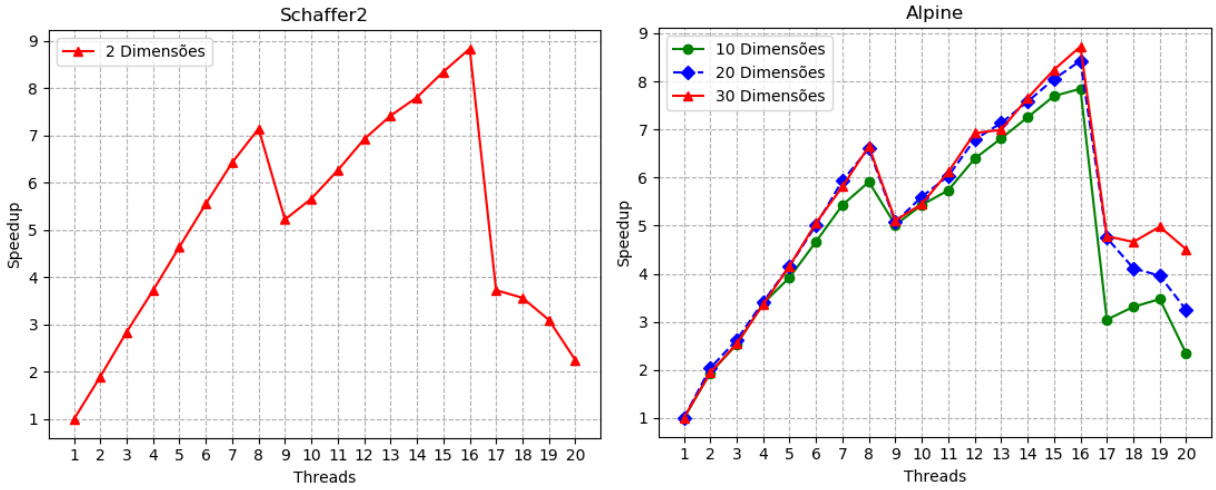
(d) f_4



(e) f_5



(f) f_6

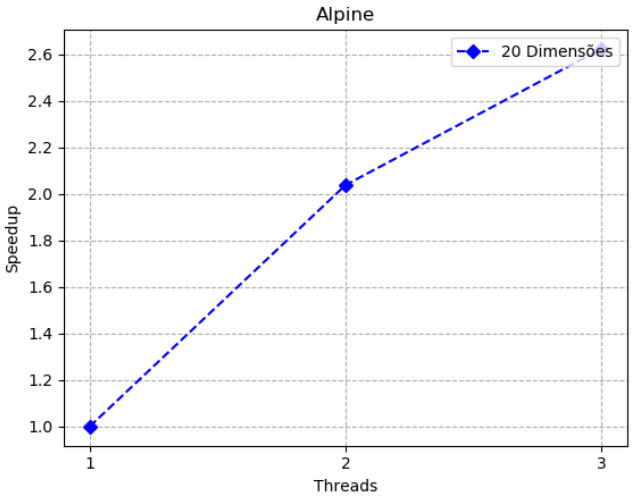


(g) f_7 (h) f_8

Fonte: Elaborado pelo autor.

A Figura 14 mostra o gráfico isolado da função *Alpine* para 20 dimensões, em um intervalo reduzido, onde é possível perceber um sutil comportamento superlinear em duas *threads*. Esse efeito é mais evidente nos resultados da análise de eficiência apresentada na próxima seção.

Figura 14 – *Speedup* da função Ackley para 20 dimensões no intervalo de uma a três *threads*.



Fonte: Elaborado pelo autor.

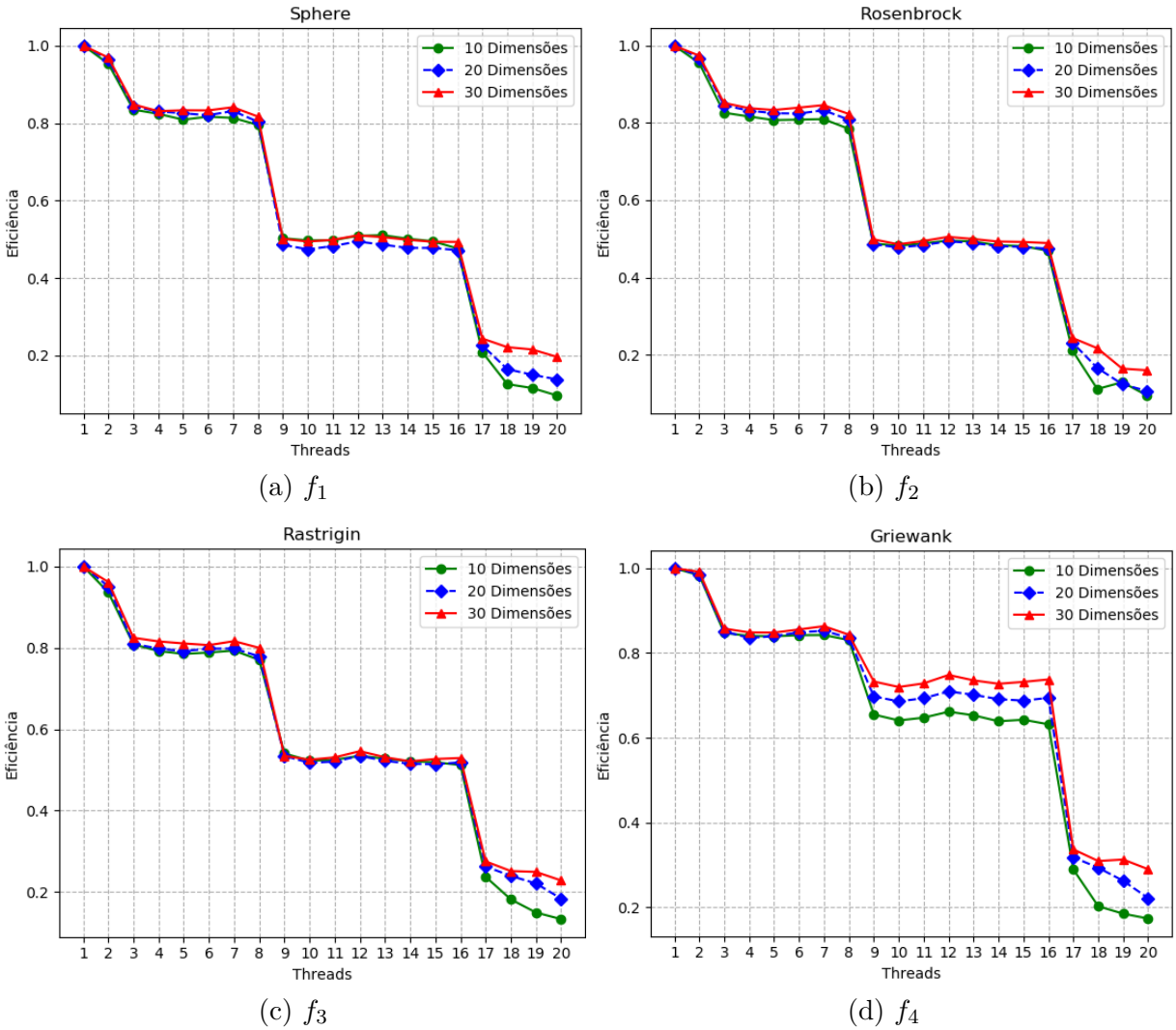
4.1.2 Eficiência

Os resultados para a análise de eficiência são exibidos na Figura 15. De acordo com o esperado, os gráficos mostram quedas acentuadas de eficiência em dois pontos distintos, para 9 e 17 *threads*. Esses decrescimentos ocorrem para números de *threads*

análogos aos dos decrescimentos presentes nos resultados de *speedup*. Isso demonstra o subaproveitamento de recursos do CPU em função da ativação do SMT e da utilização de um número de *threads* superior ao número total de núcleos da CPU.

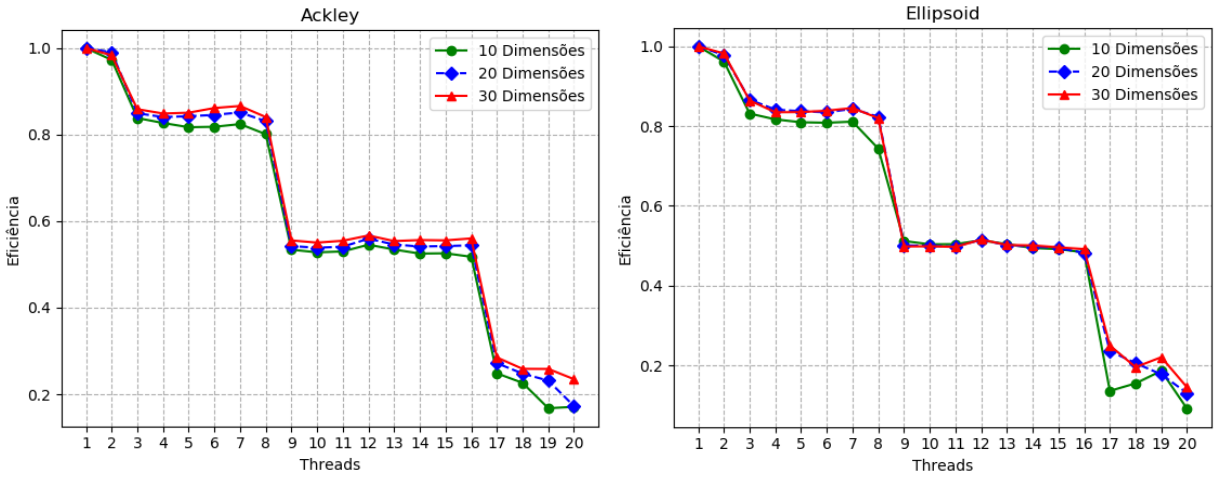
Como consequência do *speedup* superlinear, a função f_8 apresenta eficiência superior a 1 para o caso de 20 dimensões e duas *threads*. Esse é um efeito atípico e fortemente dependente da arquitetura do ambiente de execução. O trabalho de (RISTOV et al., 2016) apresenta fortes indícios de que esse efeito é consequência da utilização da memória cache pelos núcleos do processador e da otimização do algoritmo. No entanto, para afirmar com maior certeza é necessário avaliar a infraestrutura da memória cache para processadores da arquitetura *Zen*, utilizada neste estudo, no entanto, essa análise está fora do escopo deste trabalho.

Figura 15 – Resultados de eficiência.

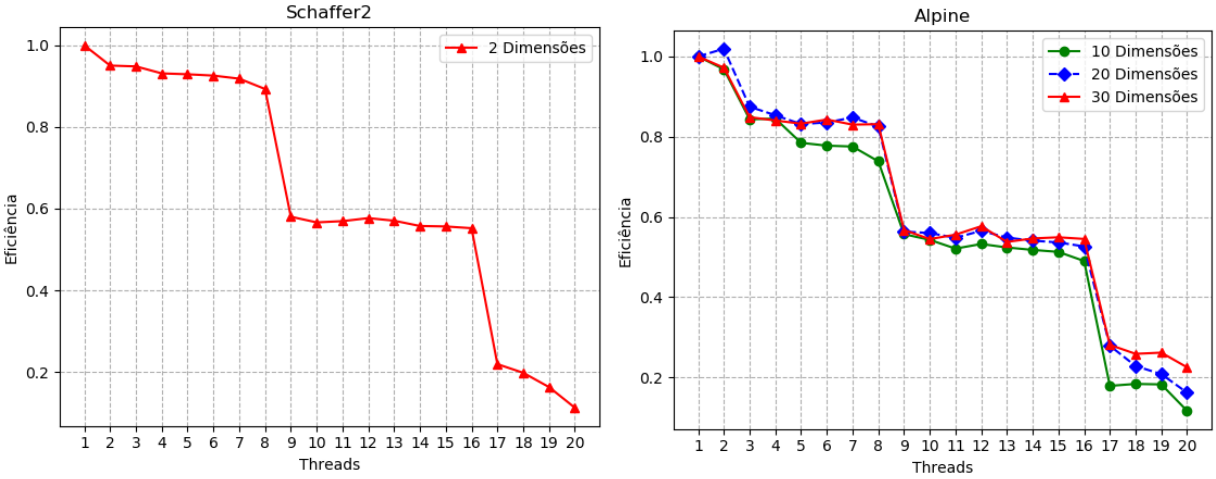


(a) f_1 (b) f_2

(c) f_3 (d) f_4



(e) f_5 (f) f_6



(g) f_7 (h) f_8

Fonte: Elaborado pelo autor.

4.2 Análise de convergência

Para este cenário, ambos os algoritmos foram executados com um número máximo de 5000 iterações, utilizando como critério de parada a identificação de soluções com *fitness* inferior a 10^{-10}, 16 *threads* e 20 partículas por enxame.

A Tabela 3 sintetiza os resultados dos testes empíricos, apresentando as médias de iterações, *fitness* e do tempo em segundos para 20 execuções. Para todos os casos de teste, o PSAPSO apresentou resultados satisfatórios. A maior discrepância de soluções foi na execução da função *Rastrigin*, principalmente para o caso de 30 dimensões. Esse comportamento pode ser explicado pelo aumento do número de partículas explorando o espaço de busca. Um comportamento semelhante poderia ser alcançado pela versão sequencial, no entanto, essa deveria utilizar um enxame com número de partículas igual a $N.p$, o que afetaria negativamente o tempo de execução. Outro fator impactado por essa característica é o número de iterações para a convergência, uma vez que o aumento do

número de partículas agilizou a convergência.

O PSAPSO apresentou tempos de execução em geral iguais ou maiores que os obtidos pelo SAPSO. Uma vez que a execução do algoritmo é síncrona, devido à utilização do modelo de comunicação, é esperado que o tempo de execução seja próximo do sequencial. Além disso, os processos de transferência de informações e criação de *threads* adicionam *overhead* à execução.

Tabela 3 – Tabela de resultados com resultados do SAPSO e do PSAPSO.

Funções	Dimensões	Métricas	SAPSO	PSAPSO
Sphere (f_1)	10	*Fitness*	8,79658E-11	**8,76336E-11**
		Iterações	999	**595**
		Tempo (seg.)	0,0587385	**0,0466954**
	20	*Fitness*	9,20776E-11	**9,02113E-11**
		Iterações	1190	**641**
		Tempo (seg.)	0,093952	**0,0781159**
	30	*Fitness*	9,4633E-11	**9,13632E-11**
		Iterações	1759	**794**
		Tempo (seg.)	0,181856	**0,131539**
Rosenbrock (f_2)	10	*Fitness*	1,40069E-06	**1,01143E-07**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,294893**	0,442026
	20	*Fitness*	0,199348	**6,64059E-07**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,430361**	0,676508
	30	*Fitness*	0,598023	**0,199332**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,559983**	0,92636
Rastrigin (f_3)	10	*Fitness*	0,106434	**0**
		Iterações	4986	**1330**
		Tempo (seg.)	0,440093	**0,182254**
	20	*Fitness*	6,8347	**5,70418E-10**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,93403**	1,58599
	30	*Fitness*	37,677	**1,73852E-09**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**1,7349**	2,80688
	10	*Fitness*	0,0757633	**0,0334565**
		Iterações	5000	**4978**
		Tempo (seg.)	**0,588591**	0,732484

Griewank (f_4)	20	*Fitness*	0,0279162	**0,0210408**
		Iterações	5000	**4846**
		Tempo (seg.)	**1,36988**	1,64752
	30	*Fitness*	0,0163615	**0,00886935**
		Iterações	4767	**4696**
		Tempo (seg.)	**2,49975**	2,90716
Ackley (f_5)	10	*Fitness*	**7,97843E-05**	0,000575653
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,480293**	0,712484
	20	*Fitness*	0,000522656	**0,000103832**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**1,0411**	1,55701
	30	*Fitness*	0,00101307	**5,29227E-05**
		Iterações	**5000**	**5000**
		Tempo (seg.)	**1,82493**	2,77175
Ellipsoid (f_6)	10	*Fitness*	8,12278E-11	**4,72445E-11**
		Iterações	**646**	2659
		Tempo (seg.)	**0,0381471**	0,218177
	20	*Fitness*	**1,44151E-10**	1,87874E-09
		Iterações	**4993**	5000
		Tempo (seg.)	**0,405057**	0,659163
	30	*Fitness*	**7,03708E-10**	5,42907E-09
		Iterações	**5000**	**5000**
		Tempo (seg.)	**0,551182**	0,909573
Schaffer N.2 (f_7)	2	*Fitness*	**4,321E-11**	4,67276E-11
		Iterações	123	**80**
		Tempo (seg.)	0,00503325	**0,00441201**
Alpine (f_8)	10	*Fitness*	1,34356E-05	**0**
		Iterações	3529	**7**
		Tempo (seg.)	0,290994	**0,00116354**
	20	*Fitness*	0,00187715	**0**
		Iterações	4752	**31**
		Tempo (seg.)	0,808748	**0,0100847**
	30	*Fitness*	11,1715	**0**
		Iterações	5000	**61**
		Tempo (seg.)	1,5244	**0,033482**

Fonte: Elaborado pelo autor.

4.2.1 Teste estatístico de Wilcoxon

O teste de estatístico de Wilcoxon foi aplicado às médias de *fitness* coletadas para os dois algoritmos. O teste foi executado utilizando o nível de significância α igual a 0,05 e um total de 22 amostras. O valor crítico, definido por tabela, foi igualado a 65 (DERRAC et al., 2011; ZAR, 2007). Foi tomado como hipótese nula a inexistência de diferença significativa entre os resultados dos dois algoritmos. Logo a hipótese alternativa afirma a existência de diferença significativa.

A tabela 4 resume os parâmetros e resultados do teste. O teste apontou os somatórios dos postos positivos R^+ igual a 225 e dos postos negativos R^- igual a 28. Portanto o T estatístico foi definido como 12. Como o valor do T estatístico foi inferior ao valor crítico, foi descartada a hipótese nula e comprovada a existência de diferença significativa entre as duas amostras de dados. Assim sendo, esse teste mostrou que a qualidade das soluções do PSAPSO superou a do SAPSO para os casos de teste analisados.

Tabela 4 – Parâmetros e resultados do teste estatístico de Wilcoxon

Parâmetros	α	0,05
	Amostras	22
	Valor crítico	65
Resultados	R^+	225
	R^+	28
	T estatístico	28

Fonte: Elaborado pelo autor.

5 Conclusão

Este trabalho apresentou a metodologia utilizada para implementar o algoritmo PSAPSO, uma variação paralela do algoritmo SAPSO, que faz uso de múltiplos enxames associados pelo paradigma mestre-escravo para estabelecer um meio de compartilhamento de informações entre núcleos de processamento. O algoritmo foi submetido a diferentes cenários de testes para demonstrar sua capacidades de investigação. A etapa de testes buscou comparar as versões sequencial e paralela, e foi corroborada a partir da aplicação do teste estatístico de Wilcoxon.

Os testes demonstraram a superioridade do PSAPSO em relação ao SAPSO. O algoritmo apresentou boa convergência para mínimos globais e agilização do processo de convergência, porém essas vantagens ocasionaram no aumento do tempo de processamento. A partir dos resultados apresentados pelo teste estatístico de Wilcoxon, é possível afirmar que os resultados obtidos não foram ocasionais, uma vez que foi assinalada a existência de diferença significativa entre as amostras avaliadas.

A utilização do processamento paralelo possibilitou uso de multiplos enxames, o que resultou no aumento do número total de partículas para exploração do espaço de busca. No entanto, em função dos custos computacionais inerentes ao processo de paralelização, o tempo mínimo de execução da versão paralela é determinado pelo tempo alcançado pela versão sequecial com tamanho de população igual a de um subenxame.

5.1 Trabalhos futuros

Levando em conta que o trabalho foi desenvolvido visando a criação de uma versão paralela e funcional, para que fosse capaz de ampliar as perspectivas de aplicação do SAPSO, deu-se pouca atenção para a otimização da implementação. Esse é um fator que pode impactar diretamente os resultados do algoritmo, demanda grande quantidade de tempo e uma análise cuidadosa do processo, dessa forma, é um tópico pertinente para trabalhos futuros.

Outro ponto de interesse observado neste trabalho foi a ocorrência do *speedup* superlinear. A análise da ocorrência desse efeito em relação à arquitetura do ambiente de execução apresenta potencial para impulsionar o desenvolvimento de novas pesquisas.

Por fim, um comparativo entre diferentes estratégias de paralelização pode se mostrar uma proposta interessante para avaliar a influência de diferentes métodos de troca de informação no processo de convergência.

Bibliografia

ABADLIA, H.; SMAIRI, N.; GHEDIRA, K. Particle swarm optimization based on dynamic island model. In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2017. p. 709–716. Citado na página 25.

ABDULLAH, E. A.; SALEH, I. A.; SAIF, O. I. A. Performance evaluation of parallel particle swarm optimization for multicore environment. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. [S.l.: s.n.], 2018. p. 81–86. Citado 3 vezes nas páginas 13, 15 e 27.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In: *AFIPS '67 Proceedings*. [S.l.: s.n.], 1967. p. 483–485. Citado na página 28.

BARLAS, G. Introduction. In: _____. *Multicore and GPU programming: An Integrated Approach*. [S.l.]: Elsevier, 2015. cap. 1, p. 1–26. Citado na página 29.

BASKAR, S.; SUGANTHAN, P. N. A novel concurrent particle swarm optimization. In: *Proceedings of the 2004 Congress on Evolutionary Computation*. [S.l.: s.n.], 2004. p. 792–796. Citado na página 13.

CHANG, J.; CHU, S.; RODDICK, J. A parallel particle swarm optimization algorithm with communication strategies. *Journal of Information Science and Engineering*, v. 21, n. 4, p. 809–818, 2005. Disponível em: <https://www.researchgate.net/publication/215744326_A_Parallel_Particle_Swarm_Optimization_Algorithm_with_Communication_Strategies>. Acesso em: 27 abr. 2019. Citado na página 13.

CICIRELLI, F. et al. Strategies for parallelizing swarm intelligence algorithms. In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2015. p. 329–336. Citado 2 vezes nas páginas 14 e 33.

CLERC, M.; KENNEDY, J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 1, p. 58–73, 2002. ISSN 1941-0026. Disponível em: <https://ieeexplore.ieee.org/document/985692>. Acesso em: 27 abr. 2019. Citado 2 vezes nas páginas 11 e 12.

DALI, N.; BOUAMAMA, S. Gpu-pso: Parallel particle swarm optimization approaches on graphical processing unit for constraint reasoning: Case of max-csps. *Procedia Computer Science*, v. 60, p. 1070–1080, 2018. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050915022796>. Acesso em: 27 abr. 2019. Citado 2 vezes nas páginas 12 e 33.

DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, v. 1, n. 1, p. 3–18, 2011. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S2210650211000034?via%3Dihub>. Acesso em: 07 jul. 2019. Citado na página 47.

FOSTER, I. Designing parallel algorithms. In: _____. *Designing and Building Parallel Programs*. [S.l.]: Addison-Wesley Longman Publishing Co., 1995. cap. 2, p. 26–82. Citado na página 23.

FREE SOFTWARE FOUNDATION. *GCC, the GNU Compiler Collection*. 2019. Disponível em: <https://gcc.gnu.org>. Acesso em: 02 jul. 2019. Citado na página 32.

FREE SOFTWARE FOUNDATION. *GDB: The GNU Project Debugger*. 2019. Disponível em: <https://www.gnu.org/software/gdb/>. Acesso em: 02 jul. 2019. Citado na página 32.

GAO, W.; LIU, S.; HUANG, L. Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique. *Communications in Nonlinear Science and Numerical Simulation*, v. 17, n. 11, p. 4316–4327, 2012. Disponível em: <https://www.sciencedirect.com/science/article/pii/S100757041200127X>. Acesso em: 27 abr. 2019. Citado na página 12.

GBENGA, D. E.; RAMLAN, E. I. Understanding the limitations of particle swarm algorithm for dynamic optimization tasks: A survey towards the singularity of pso for swarm robotic applications. *ACM Computing Surveys*, v. 49, n. 8, p. 1–25, 2016. ISSN 1557-7341. Disponível em: <https://dl.acm.org/citation.cfm?id=2906150>. Acesso em: 26 abr. 2019. Citado 2 vezes nas páginas 11 e 12.

GüLCü, S.; KODAZ, H. A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization. *Engineering Applications of Artificial Intelligence*, v. 45, p. 33–45, 2015. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0952197615001359>. Acesso em: 27 abr. 2019. Citado 6 vezes nas páginas 13, 26, 31, 34, 37 e 39.

HAN, F.; LIU, Q. A diversity-guided hybrid particle swarm optimization based on gradient search. *Neurocomputing*, v. 137, p. 234 – 240, 2014. ISSN 0925-2312. Advanced Intelligent Computing Theories and MethodologiesSelected papers from the 2012 Eighth International Conference on Intelligent Computing (ICIC 2012). Citado na página 14.

JONG, K. A. D. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Tese (Doutorado) — University of Michigan, Ann Arbor, MI, USA, 1975. AAI7609381. Citado 2 vezes nas páginas 16 e 37.

KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995. p. 1942–1948. Citado 3 vezes nas páginas 11, 12 e 18.

KENNEDY, J.; EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In: *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*. [S.l.: s.n.], 1997. p. 4104–4108. Citado na página 11.

KUBUNTU DEVS. *Kubuntu | Friendly Computing*. 2019. Disponível em: <https://kubuntu.org>. Acesso em: 02 jul. 2019. Citado na página 32.

LALWANI, S. et al. A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering*, v. 44, n. 4, p. 2899–2923, 2019. Disponível em:

<https://link.springer.com/article/10.1007%2Fs13369-018-03713-6>. Acesso em: 07 jul. 2019. Citado na página 25.

LI, X.; CLERC, M. Swarm intelligence. In: _____. *Handbook of Metaheuristics*. 3. ed. Montreal: Springer, 2019. cap. 11, p. 353–384. Citado na página 12.

MATHWORKS. *MATLAB - MathWorks - MATLAB Simulink*. 2019. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 02 jul. 2019. Citado na página 32.

MENDES, R.; J.KENNEDY; NEVES, J. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, v. 8, n. 3, p. 204–210, 2004. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1304843>. Acesso em: 27 abr. 2019. Citado na página 12.

MIRANDA, V.; FONSECA, N. Epso - evolutionary particle swarm optimization, a new algorithm with applications in power systems. In: *IEEE/PES Transmission and Distribution Conference and Exhibition*. [S.l.: s.n.], 2002. p. 745–750. Citado na página 11.

MUSSI, L.; DAOLIO, F.; CAGNONI, S. Evaluation of parallel particle swarm optimization algorithms within the cuda™ architecture. *Information Sciences*, v. 181, n. 20, p. 4642–4657, 2011. ISSN 0020-0255. Citado na página 25.

NOEL, M. M. A new gradient based particle swarm optimization algorithm for accurate computation of global minimum. *Applied Soft Computing*, v. 12, n. 1, p. 353 – 359, 2012. ISSN 1568-4946. Citado 2 vezes nas páginas 14 e 20.

OPENMP. *The OpenMP API specification for parallel programming*. 2019. Disponível em: <https://www.openmp.org>. Acesso em: 02 jul. 2019. Citado na página 32.

OZCAN, E.; MOHAN, C. K. Particle swarm optimization: Surfing the waves. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*. [S.l.: s.n.], 1999. p. 1939–1944. Citado na página 13.

RISTOV, S. et al. Superlinear speedup in hpc systems: why and when? In: *Proceedings of the Federated Conference on Computer Science and Information Systems*. [S.l.: s.n.], 2016. p. 889–898. Citado 2 vezes nas páginas 37 e 43.

SANTOS. *The SAPSO algorithm developed in MATLAB*. 2018. Disponível em: <https://github.com/regicsf2010/SAPSO>. Acesso em: 02 jul. 2019. Citado na página 32.

SANTOS, R. et al. A semi-autonomous particle swarm optimizer based on gradient information and diversity control for global optimization. *Applied Soft Computing*, v. 69, p. 330–343, 2018. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494618302187>. Acesso em: 27 abr. 2019. Citado 9 vezes nas páginas 11, 14, 15, 16, 19, 23, 37, 38 e 39.

SHANG, Y.; QIU, Y. A note on the extended rosenbrock function. *Evolutionary Computation*, v. 14, n. 1, p. 119–126, 2006. Citado na página 37.

SHI, T.; EBERHART, R. C. Empirical study of particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*. [S.l.: s.n.], 1999. p. 1945–1950. Citado na página 13.

SÖRENSEN, K. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, v. 22, n. 1, p. 3–18, 2015. ISSN 1475-3995. Citado na página 14.

STANDARD C++ FOUNDATION. *Standard C++*. 2019. Disponível em: <https://isocpp.org>. Acesso em: 02 jul. 2019. Citado na página 32.

SZCZEPANSKI, R.; ERWINSKI, K.; PAPROCKI, M. Accelerating pso based feedrate optimization for nurbs toolpaths using parallel computation with openmp. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. [S.l.: s.n.], 2017. p. 431–436. Citado na página 13.

THE QT COMPANY. *Libraries APIs, Tools and IDE | Qt*. 2019. Disponível em: <https://www.qt.io/qt-features-libraries-apis-tools-and-ide/#ide>. Acesso em: 02 jul. 2019. Citado na página 32.

TIAN, D.; SHI, Z. Mpso: Modified particle swarm optimization and its applications. *Swarm and Evolutionary Computation*, v. 41, p. 49–68, 2018. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S2210650217307137>. Acesso em: 27 abr. 2019. Citado na página 12.

VESTERSTRØM, J.; RIGET, J. A diversity-guided particle swarm optimizer - the arpso. *EVALife Technical Report*, n. 2002-02, 2002. Citado 2 vezes nas páginas 14 e 20.

VILANOVA, L.; AMIT, N.; ETSION, Y. Using smt to accelerate nested virtualization. In: *Proceedings of the 46th International Symposium on Computer Architecture*. [S.l.: s.n.], 2019. p. 750–761. Citado na página 40.

WACHOWIAK, M.; TIMSON, M.; DUVAL, D. Adaptive particle swarm optimization with heterogeneous multicore parallelism and gpu acceleration. *IEEE Transactions on Parallel and Distributed Systems*, v. 28, n. 10, p. 2784–2793, 2017. Disponível em: <https://ieeexplore.ieee.org/document/7886331>. Acesso em: 27 abr. 2019. Citado na página 13.

WANG, S.; SU, L.; ZHANG, J. Mpi based pso algorithm for the optimization problem in micro-grid energy management system. In: *2017 Chinese Automation Congress (CAC)*. [S.l.: s.n.], 2017. p. 4479–4483. Citado na página 14.

ZAR, J. H. *Biostatistical Analysis (5th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2007. ISBN 0131008463. Citado na página 47.