Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

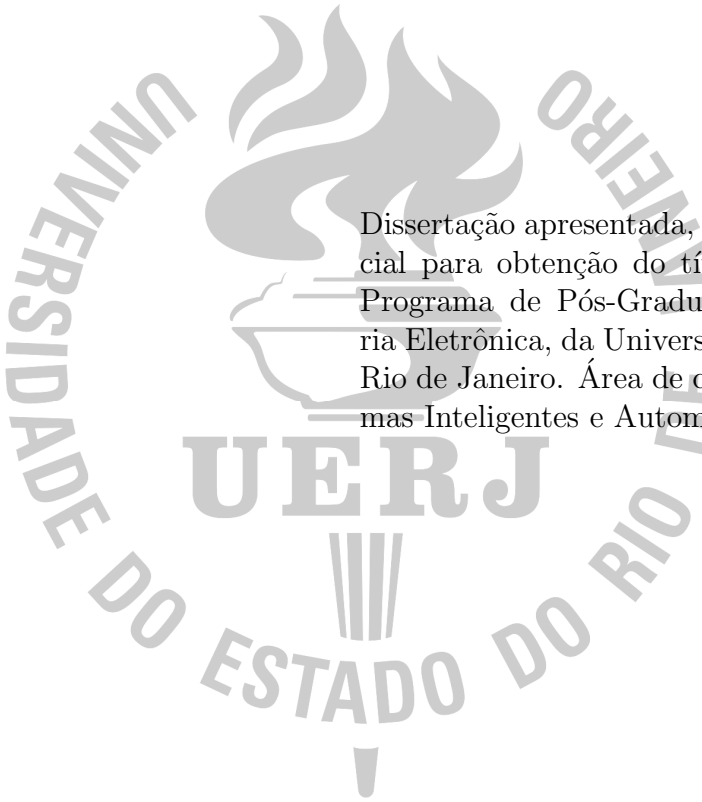
Alberto de Carvalho Passos

Implementação paralela do algoritmo de otimização por enxame de partículas em uma plataforma multiprocessada com rede intrachip

Rio de Janeiro
2024

Alberto de Carvalho Passos

Implementação paralela do algoritmo de otimização por enxame de partículas em uma plataforma multiprocessada com rede intrachip



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof.^{a} Dr.^{a} Luiza de Macedo Mourelle
Orientadora: Prof.^{a} Dr.^{a} Nadia Nedjah

Rio de Janeiro
2024

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

P289 Passos, Alberto de Carvalho.
Implementação paralela do algoritmo de otimização por enxame de partículas em uma plataforma multiprocessada com rede intrachip / Alberto de Carvalho Passos. – 2024.
104 f.

Orientadoras: Luiza de Macedo Mourelle, Nadia Nedjah.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia eletrônica - Teses. 2. Inteligência coletiva - Teses. 3. Algorítmos paralelos - Teses. 4. Sistemas embarcados (Computadores) - Teses. 5. Sistemas programáveis em chip - Teses. I. Mourelle, Luiza de Macedo. II. Nedjah, Nadia. III. Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia. IV. Título.

CDU 004.896

Bibliotecária: Júlia Vieira – CRB7/6022

Assinatura _____ Data _____

Alberto de Carvalho Passos

Implementação paralela do algoritmo de otimização por enxame de partículas em uma plataforma multiprocessada com rede intrachip

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em: 27/03/2024

Banca Examinadora:

Prof.ª Dr.ª Luiza de Macedo Mourelle (Orientadora)
Faculdade de Engenharia, UERJ

Prof.ª Dr.ª Nadia Nedjah (Orientadora)
Faculdade de Engenharia, UERJ

Prof. Dr.ª Maria Clícia Stelling de Castro
Instituto de Matemática e Estatística, UERJ

Prof. Dr. Paulo Victor Rodrigues de Carvalho
Programa de Pós-Graduação em Informática, UFRJ

Rio de Janeiro
2024

Faça as coisas o mais simples que você puder,
porém não se restrinja às mais simples.

Albert Einstein

RESUMO

Alberto de Carvalho Passos. *Implementação paralela do algoritmo de otimização por enxame de partículas em uma plataforma multiprocessada com rede intrachip* . 2024. 104f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

Nos últimos anos surgiu a necessidade de resolver problemas complexos em várias áreas do conhecimento, como mineração de dados, otimização combinatória, sistemas de energia, processamento de sinais, reconhecimento de padrões, aprendizado de máquina e robótica. A característica chave desses problemas é a sua intensidade computacional, especialmente em termos de tempo de execução. Para acelerar o processo de resolução de problemas, foram desenvolvidos algoritmos bioinspirados, que visam simular o comportamento encontrado em sistemas biológicos, como organismos vivos e ecossistemas, para resolver eficientemente problemas complexos. Exemplos desses algoritmos incluem Otimização por Enxame de Partículas, Otimização por Colônia de Formigas, Colônia Artificial de Abelhas e Busca Cuckoo. Este trabalho tem como objetivo obter uma implementação paralela do algoritmo de Otimização por Enxame de Partículas utilizando um Sistema Embutido Multiprocessado com Rede Intrachip. As estratégias de paralelização que empregamos são baseadas nos algoritmos PSO Paralelo (Parallel PSO - PPSO) e PSO Cooperativo (Cooperative Particle Swarm Optimizer - CPSO), utilizando as topologias Mestre-trabalhador, Anel e Malha 2D. Com base no tempo de execução obtido por cada algoritmo paralelo e cada topologia empregada durante as simulações, será possível identificar qual estratégia de paralelização oferece o melhor desempenho, bem como o número de processadores necessários. Os resultados, quando comparados com a versão serial do algoritmo de Otimização por Enxame de Partículas, são promissores.

Palavras-chave: Otimização por enxame de partículas; Algoritmos paralelos; Sistema embutido multiprocessado; Redes intrachip.

ABSTRACT

PASSOS, Alberto De Carvalho. *Parallel Implementation of the Particle Swarm Optimization Algorithm on a Multiprocessor Embedded System with Network-on-Chip.* 2024. 104f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 202x.

In recent years, with technological advancements, the need to solve complex problems has emerged in various areas of knowledge, such as data mining, combinatorial optimization, power systems, signal processing, pattern recognition, machine learning, and robotics. The key characteristic of these problems is their computational intensity, particularly in terms of execution time. In order to accelerate the problem-solving process, bio-inspired algorithms have been developed, which aim to simulate the behavior found in biological systems, such as living organisms and ecosystems, to efficiently solve complex problems. Examples of these algorithms include Particle Swarm Optimization, Ant Colony Optimization, Artificial Bee Colony, and Cuckoo Search. This work aims to obtain a parallel implementation of the Particle Swarm Optimization algorithm using a Multiprocessor Embedded System with Network-on-Chip. The parallelization strategies we employ are based on the Parallel Particle Swarm Optimization (PPSO) and Cooperative Parallel Particle Swarm Optimization algorithms (CPSO), using master-slave, ring, and 2D grid topologies. Based on the execution time obtained by each parallel algorithm and each employed topology during the simulations, it will be possible to identify which parallelization strategy provides the best performance, as well as the number of processors required. The results, when compared to the serial version of the Particle Swarm Optimization algorithm, are promising.

Keywords: Particle swarm optimization; Parallel algorithms; Multiprocessor embedded system; Network-on-chip.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Arquitecture
DMA	Direct Memory Access
FPGA	Field-Programmable Gate Arrays
GPU	Graphics Processing Unit
HEMPS	Hermes Multiprocessor System on Chip
MEMPHIS	Many-core Modeling Platform for Heterogenous Systems on Chip
PE	Processing Element
PSO	Particle Swarm Optimization
SoC	System on Chip
VHDL	Very High Speed Integrated Circuit Hardware Description Language

SUMÁRIO

INTRODUÇÃO

ULTIMAMENTE a procura por otimização numérica tem se tornado cada vez mais frequente. Praticamente todas as áreas do conhecimento necessitam e utilizam esta ferramenta. Na aerodinâmica, por exemplo, pode ser utilizada para proporcionar redução do consumo de combustível da aeronave, que é conseguido através da otimização no design das asas. Na área de finanças, o portfólio de um determinado fundo pode utilizar a otimização visando maximizar seu lucro. Empresas de logística tem a possibilidade de otimizar suas rotas de transportes diminuindo custos e áreas como a Química pode se beneficiar ao otimizar experimentos, ao encontrar, por exemplo, as condições ideais de reação que maximizam o rendimento de um produto desejado.

O conceito de inteligência de enxame, se baseia no comportamento coletivo de organismos, como pássaros, abelhas e formigas, tendo por princípio o comportamento global de determinado enxame bem como o individual. Neste trabalho, concentramos nossa atenção na otimização por enxame de partículas *(Particle Swarm Optimization - PSO)*. Esta escolha se deu devido à sua ampla utilização principalmente motivados pela sua eficiência, adaptabilidade e a possibilidade de ser paralelizado sem muita complexidade.

O foco principal, neste trabalho, é obter uma execução eficiente e rápida do PSO, explorando o paralelismo inerente a este cálculo. Na busca do que pode ser paralelizado precisamos entender o mecanismo que rege este algoritmo. A base fundamental é a criação de um enxame de partículas, que são distribuídas em determinado espaço de busca. Cada partícula passa a ter sua velocidade e deslocamento regidas por equações que tem como objetivo a otimização de uma determinada função. Esta execução tem se tornado intensiva computacionalmente devido à necessidade de lidar com grandes quantidades de dados ao resolver problemas envolvendo numerosas partículas ou dimensões. O desempenho alcançado pelo PSO, nessas condições, pode ser afetado, pois o tempo de processamento para alcançar o resultado pode aumentar significativamente.

A paralelização do PSO vem sendo explorada com o intuito de oferecer uma execução mais rápida. Neste caso, temos soluções em software utilizando CUDA, OPENMP, MPI e em hardware, fazendo uso de FPGA para implementar blocos dedicados ao cálculo do PSO. Neste trabalho, orientamos a solução para um sistema embutido multiprocessado escalável, baseado em rede intrachip.

Abordaremos os algoritmos PSO Paralelo (Parallel PSO-PPSO) e o PSO Cooperativo (Cooperative Particle Swarm Optimization - CPSO) executando em uma arquitetura multiprocessada escalável com rede intrachip denominada MEMPHIS. Outro ponto de atenção é com relação ao tempo gasto em comunicação entre os processadores. As simulações serão feitas considerando 3 topologias de comunicação: Mestre-trabalhador, Anel e Malha 2D. Nosso objetivo será identificar a relação entre o número de processadores e o tempo de execução. Estas são informações relevantes principalmente se a intensão é a implementação em *hardware* como uma FPGA. Quanto menos processadores utilizados, menor a área de FPGA utilizada e consequentemente menor o custo.

O conteúdo desta dissertação está organizado em seis capítulos. O Capítulo 1 apresenta o conceito de otimização e introduz o PSO como uma solução viável e já bastante utilizado quando necessitamos otimizar processos. São abordados os algoritmos Global Best PSO e Local Best PSO, assim como os parâmetros que fazem parte de suas equações.

O Capítulo 2 apresenta as estratégias de paralelização, detalhando três diferentes arranjos, Mestre-trabalhador, Anel e Malha 2D, que determinam como ocorrerá a troca de informações entre processadores. Em seguida são apresentados os algoritmos paralelos síncronos e assíncronos do PSO e o algoritmo paralelo cooperativo (CPSO), este utilizado em nossos experimentos.

O Capítulo 3 apresenta os trabalhos relacionados que foram utilizados como referência para este trabalho. O estudo foi organizado em três seções que abordam, respectivamente, a paralelização baseada em CPU, GPU e *hardware*.

O Capítulo 4 apresenta o ambiente de simulação MEMPHIS (*Many-core Modeling Platform for Heterogeneous* SoCs). Este ambiente permite a criação de sistemas baseados em processamentos múltiplos utilizando um único chip, e empregando uma rede intrachip como canal de comunicação.

O Capítulo 5 apresenta os resultados experimentais. Para os experimentos, variamos o número de processadores em 3 diferentes tipos de topologias de comunicação, Mestre-trabalhador, Anel e Malha 2D.

O Capítulo 6 apresenta as principais conclusões obtidas e sugestões para trabalhos futuros. Procuramos, aqui, destacar o *speedup* obtido para as diferentes configurações e o impacto que o tempo de comunicação entre os processadores causa. Alguns aspectos da MEMPHIS são observados, os quais impõem algumas restrições no seu uso.

Capítulo 1

OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

ESTE capítulo apresenta o algoritmo de otimização por enxame de partículas (Particle Swarm Optimization - PSO) apresentado por (KENNEDY; EBERHART, 1995). Trata-se de uma abordagem heurística que utiliza um enxame de partículas que, por meio de comportamentos sociais e cognitivos, buscam a solução de acordo com o objetivo definido. As partículas se movem pelo espaço de busca à procura da solução ótima. Cada uma representa uma solução candidata, tendo uma posição e velocidade associadas. Durante a otimização, as partículas são influenciadas por duas informações: sua melhor posição pessoal, associada ao seu comportamento cognitivo, e a melhor posição global encontrada pelo enxame, associada a seu comportamento social. A Seção 1.1 faz uma apresentação inicial ao PSO, introduzindo as equações que regem sua dinâmica. A Seção 1.2 apresenta o algoritmo do PSO baseado no melhor global. A Seção 1.3 apresenta o algoritmo do PSO baseado no melhor local. A Seção 1.4 aborda os principais parâmetros utilizados nestes algoritmos. A Seção 1.5 apresenta as considerações finais.

1.1 Introdução ao PSO

O PSO é frequentemente utilizado para resolver problemas de otimização, que envolvem funções matemáticas e questões de engenharia, devido a sua simplicidade, eficiência e capacidade de encontrar soluções próximas ao ótimo global. Este algoritmo, aproveita a colaboração e interação entre as partículas para descobrir soluções aprimoradas ao longo do tempo. Conforme originalmente proposto por (ENGELBRECHT, 2005), o movimento de cada partícula é definido pelas Equações (1) e (2):

$$v_{k+1} = v_k + c_1 r_1 (pbest_k - x_k) + c_2 r_2 (gbest - x_k), \quad (1)$$

$$x_{k+1} = x_k + v_{k+1}, \quad (2)$$

onde v_{k+1} e x_{k+1} são respectivamente a velocidade e a posição da partícula no instante $k+1$, c_1 e c_2 são os coeficientes de aprendizado. Estes controlam a influência da melhor posição individual *pbest* e da melhor posição global *gbest*, respectivamente, determinando a influência da exploração individual e global no processo de otimização. r_1 e r_2 são valores aleatórios entre 0 e 1, que promovem diversidade na busca evitando que as partículas fiquem presas em ótimos locais. x_k representa a posição atual da partícula. *pbest*, que inicialmente possui os mesmos valores das coordenadas iniciais da partícula, refere-se à melhor posição encontrada pela partícula até o momento, correspondendo ao componente cognitivo. *gbest* representa a melhor posição encontrada pelo enxame até aquele momento, correspondendo ao componente social. Os vetores que atuam na definição da nova posição da partícula estão ilustrados na Figura 1.

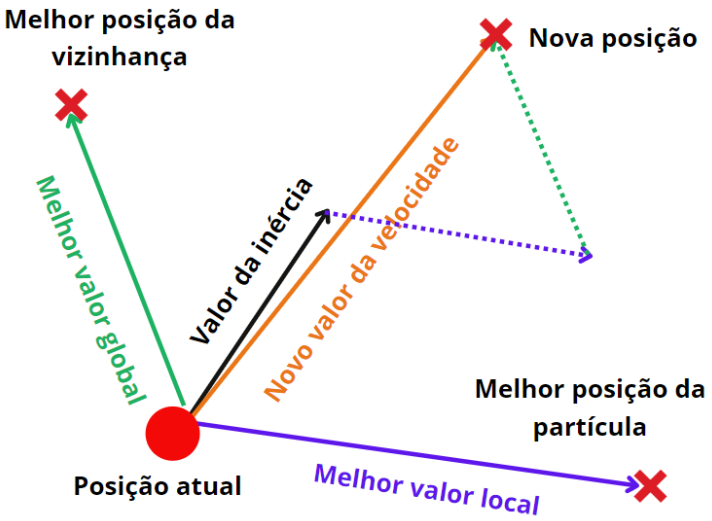


Figura 1: Cálculo da nova posição da partícula

A posição final da partícula é determinada pela sua posição atual mais o valor de deslocamento conforme a Equação (2). Uma importante contribuição foi dada por (SHI; EBERHART, 1999), que introduziu a influência da inércia w, que controla a influência da velocidade anterior, com valores entre 0,4 e 1,0. Com a inclusão desta variável podemos reescrever a equação da velocidade conforme a Equação 3:

$$v_{k+1} = w v_k + c_1 r_1 (pbest_k - x_k) + c_2 r_2 (gbest - x_k). \quad (3)$$

Outra sugestão de alteração para o algoritmo do PSO foi feita por (CLERC, 1999) com a introdução do fator de constrição. Este parâmetro é utilizado no PSO para ajustar a velocidade das partículas durante o processo de otimização com o objetivo de melhorar a estabilidade e a convergência. O fator de constrição é introduzido para garantir que a velocidade das partículas permaneça dentro de limites específicos, evitando variações bruscas e contribuindo para uma convergência mais estável do algoritmo. Com a inclusão do fator de constrição definido por χ, podemos reescrever a equação da velocidade conforme a Equação 4:

$$v_{k+1} = \chi[wv_k + c_1 r_1(pbest_k - x_k) + c_2 r_2(gbest - x_k)]. \tag{4}$$

1.2 Algoritmo Global Best PSO

São diversas as topologias de vizinhança possíveis quando utilizamos o PSO. Abordaremos inicialmente a topologia chamada Gbest, onde cada partícula influencia todas as outras, ou seja, se trata de uma topologia totalmente conectada Figura 2.

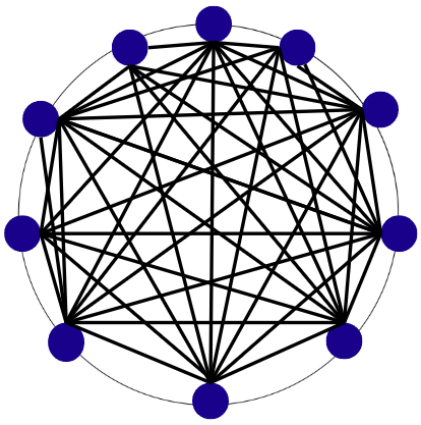


Figura 2: Topologia totalmente conectada

O Algoritmo 1 apresenta as etapas relativas ao Global Best PSO. Primeiramente ocorre a inicialização dos parâmetros e variáveis definidos por w para o valor de inércia, c_1 e c_2 que são respectivamente os coeficientes de aprendizado cognitivo e social e as variáveis r_1 e r_2 que assumirão valores aleatórios entre 0 e 1. O número de partículas é definido por *npart*. A criação do enxame se dá pela distribuição aleatória das partículas no espaço de busca. A função de aptidão, correspondente à função a ser otimizada, é aplicada a cada partícula, a cada iteração, enquanto os valores de *pbest* e *gbest* são

atualizados. No algoritmo consideramos a vizinhança de cada partícula como sendo todas as demais partículas do enxame. Logo, o valor de melhor local de determinada partícula é sempre comparado com o melhor valor encontrado pelas demais partículas do enxame. O próximo passo é atualizar a velocidade e posição de cada partícula conforme as equações 3 e 2 respectivamente. A condição de parada pode ser baseada no número máximo de iterações, em um limite de precisão ou em uma melhoria mínima na solução.

Algoritmo 1 Global Best PSO

```
1: inicializar w, c1, c2, npart;
2: Criar enxame de partículas;
3: repita
4:    para i := 1 → npart faça
5:       Calcular aptidão da partícula;
6:       se aptidão < pbest_i então
7:          atualizar pbest_i
8:       fim se
9:       se pbest_i <= gbest então
10:         atualizar gbest;
11:      fim se
12:      atualizar velocidade da partícula usando Eq. (3);
13:      atualizar posição da partícula usando Eq. (2);
14:   fim para
15: até condição de parada
16: retorna gbest
```

1.3 Algoritmo Local Best PSO

Em Algoritmo 2, apresentamos as etapas de otimização requeridas ao Local Best PSO . A principal vantagem do Local Best PSO é que ele converge mais lentamente, aumentando a chance de encontrar um ótimo global. As partículas tendem a se mover mais em direção às soluções dentro da vizinhança local. Ao evitar uma rápida convergência prematura para uma possível solução local ótima, o Local Best PSO permite que sejam exploradas outras áreas do espaço de busca que podem conter soluções melhores evitando assim a convergência prematura para um mínimo local.(ENGELBRECHT, 2006).

No algoritmo Local Best PSO, descrito pelo Algoritmo 2, a ideia é similar ao Global Best PSO, sendo que a diferença está na vizinhança que influencia cada partícula. Segundo (KENNEDY; MENDES, 2002), cada partícula teria um número fixo de vizinhos ao seu redor, que poderiam ser dois, ou seja, um do lado esquerdo e outro do lado direito, caracterizando uma estrutura ou topologia em Anel conforme mostra a Figura 3. Neste

tipo de topologia a troca de informações se dará apenas entre os vizinhos, que desta forma sofreriam mútua influência conforme seus melhores resultados encontrados.

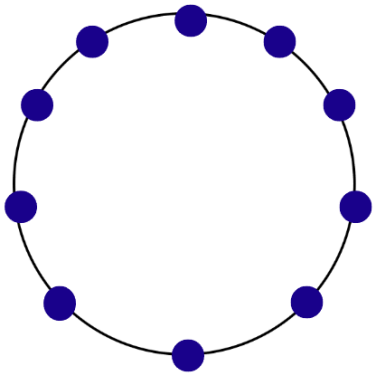


Figura 3: Topologia em Anel

Algoritmo 2 Local Best PSO

```
1: inicializar w, c1, c2, npart;
2: Criar enxame de partículas;
3: repita
4:    para i := 1 → npart faça
5:       Calcular aptidão da partícula;
6:       se aptidão < pbest_i então
7:          atualizar pbest_i
8:       fim se
9:       se pbest_i <= lbest então
10:         atualizar lbest;
11:      fim se
12:      atualizar velocidade da partícula usando Eq. (1);
13:      atualizar posição da partícula usando Eq. (2);
14:   fim para
15: até condição de parada
16: retorna lbest
```

No algoritmo Local Best PSO, a variável *lbest* é influenciada conforme a maneira como as partículas interagem em relação à sua vizinhança local. Além da vizinhança que se carateriza pela comunicação em Anel podemos citar as topologias Malha 2D ou Von Neumann e Mestre-trabalhador, conforme ilustrado na Figura 4. São topologias de comunicação amplamente utilizadas quando tratamos com paralelização.

A topologia Malha 2D é organizada em uma estrutura de grade bidimensional, onde cada partícula tem um vizinho na direita, esquerda, cima e baixo. Este comportamento possibilita uma propagação da informação superior ao da topologia Local Best.

Na topologia Mestre-trabalhador, as interações entre as partículas são centralizadas em torno do líder. Cada partícula é influenciada pelas informações provenientes do líder, e as atualizações de posição e velocidade são realizadas com base nessa interação.

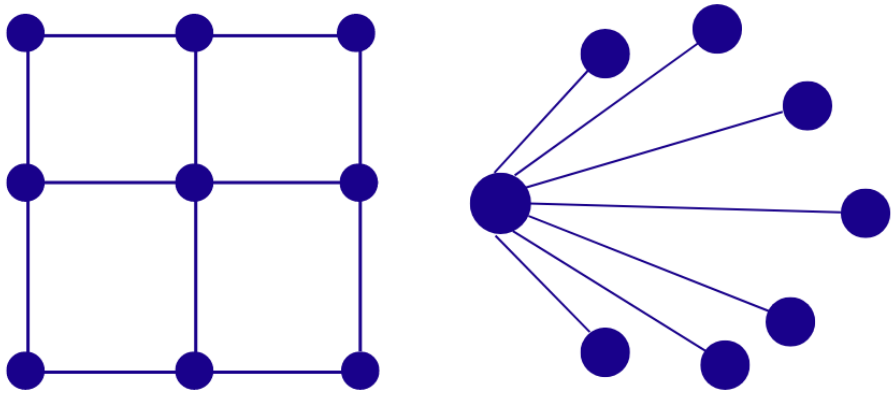


Figura 4: Topologias utilizadas no PSO

Outro tipo de topologia é a do tipo Árvore ou hierárquica conforme a Figura 5. As partículas são organizadas em uma hierarquia, onde cada uma tem um pai e possivelmente filhos, formando uma estrutura semelhante a uma árvore. A partícula que está mais acima (pai) influencia a partícula conectada no nó imediatamente abaixo. Se uma partícula filha encontra uma posição melhor, a partícula pai atualiza sua posição conforme este valor. Pode-se aumentar a complexidade desta estrutura combinando topologias do tipo de Anel ou a Malha 2D, levando a diferentes tipos de interações conforme os níveis da hierarquia.

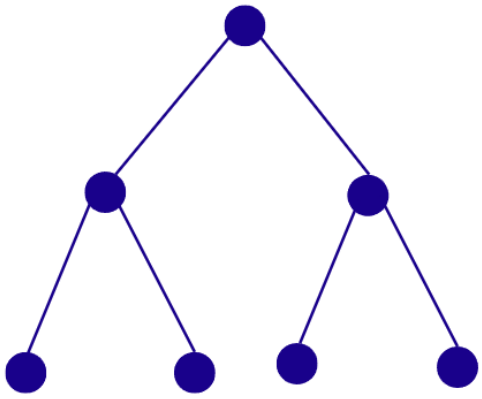


Figura 5: Topologia em árvore ou hierárquica

A eficácia quando alteramos o tipo de topologia utilizada pode variar e depende também dos parâmetros utilizados como: fator de inércia e os coeficientes cognitivo e social.

No trabalho apresentado por (KENNEDY, 1999), foram realizados experimentos utilizando funções variadas e diferentes tipos de topologias. O desempenho variou conforme a topologia e função utilizadas, sendo inconclusivo afirmar qual oferece a melhor solução.

Na tentativa de aproveitar o melhor de cada topologia é que surgiu a proposta da topologia dinâmica apresentada por (RICHARDS; VENTURA, 2004). A ideia é alterar a forma de conexão entre as partículas durante a execução do algoritmo. Este processo ocorreria de forma coordenada na tentativa de se conseguir uma melhor exploração do espaço de busca conforme o andamento do algoritmo. No início do processo de otimização pode-se permitir uma comunicação mais ampla entre as partículas e para evitar que todas convirjam rapidamente para uma mesma solução, que pode não ser a melhor, pode-se reduzir o número de comunicações com as partículas vizinhas. Esta redução pode ser obtida ao utilizarmos uma vizinhança do tipo anel. Desta forma, as partículas conseguem explorar por mais tempo e de forma mais abrangente o espaço de busca.

1.4 Parâmetros do Gbest PSO e do Lbest PSO

Os parâmetros utilizados têm impacto significativo no desempenho e no resultado durante o processo de otimização, sendo importante buscar o equilíbrio destes valores. O fator de inércia w conforme (SHI; EBERHART, 1999) controla a influência da velocidade anterior da partícula na sua próxima velocidade. Um valor alto faz com que a partícula continue se movendo rapidamente na mesma direção sem explorar melhor o espaço de busca e solução, possivelmente melhores, nas proximidades podem passar despercebidas. Neste trabalho também é levantado a proposta da utilização da variação linear para este parâmetro. A ideia é diminuir o fator de inércia ao longo das iterações, para que as partículas diminuam suas velocidades ao longo da execução do algoritmo, e explorem melhor o espaço de busca.

O número de partículas pode permitir uma busca mais ou menos ampla do espaço de busca. Quanto maior número de partículas melhor será a cobertura na busca do espaço de solução. Neste caso, também aumenta-se o poder computacional necessário bem como o custo de implementação.

Os coeficientes de Aprendizado c_1 e c_2 determinam a influência das experiências individuais *pbest* e globais *gbest* na atualização da velocidade das partículas. A importância relativa dos coeficientes de aprendizado individual e global depende do problema específico que está sendo otimizado e da dinâmica do espaço de busca. Em geral, um

equilíbrio entre estes parâmetros é desejável. Os valores de r_1 e r_2 utilizados na equação do PSO, permitem introduzir um fator aleatório na atualização das velocidades de cada partícula. Esse fator é incluído utilizando-se de números aleatórios gerados entre 0 e 1. Estes números são multiplicados pelos termos relacionados respectivamente a *pbest* e *gbest*, e possibilitam incluir um grau de aleatoriedade no peso destes fatores, o que pode ajudar na exploração do espaço de busca (CLERC, 2010).

O critério de parada determina quando o algoritmo PSO deve terminar. Ao ajustar esses parâmetros, deve-se propiciar ao algoritmo a exploração adequada do espaço de busca e garantir que o algoritmo alcance uma solução aceitável dentro de um tempo computacional razoável.

1.5 Considerações Finais

Neste capítulo é apresentado uma introdução ao PSO, que tem como objetivo a otimização de uma função, encontrando seus pontos de máximo ou mínimo. O processo ocorre com a utilização de partículas, que são dispostas no espaço aonde se encontram as possíveis soluções. Essas partículas interagem de forma iterativa, através da colaboração e comunicação de forma a encontrar as melhores soluções no espaço de busca. Foram apresentado os vetores que participam da movimentação das partículas conhecidos como componentes cognitivo e social bem como os demais parâmetros que fazem parte das equações fundamentais de deslocamento e velocidade das partículas. Também foram abordados as diferentes formas de comunicação, Local Best PSO e Global Best PSO, entre as partículas da vizinhança através do estudo das topologias de comunicação. Ao final do capítulo foi realizado um apanhado geral sobre vários tipos de topologia Local de vizinhança que podem ser utilizados. No capítulo seguinte abordaremos as estratégias e topologias que utilizaremos em nossos experimentos.

Capítulo 2

ESTRATÉGIAS DE PARALELIZAÇÃO

ESTE capítulo apresenta as propostas de paralelização para o PSO e o porque dessa necessidade. Afinal, qual o motivo em tornar mais rápido a execução deste algoritmo? A questão é que nos últimos anos, com os avanços tecnológicos, surgiu a necessidade de resolver problemas complexos em diversas áreas do conhecimento, como mineração de dados, otimização combinatória, sistemas de energia, processamento de sinais, reconhecimento de padrões, aprendizado de máquina e robótica. A característica chave desses problemas é a intensidade computacional, especialmente em termos de tempo de execução. Neste sentido, surgiu a necessidade da criação de métodos para acelerar este processamento. A proposta é aproveitar a vantagem do paralelismo, que, entre outras possibilidades, facilita a atualização simultânea de múltiplas partículas, resultando em um tempo menor para atingir uma solução. Através do paralelismo, também é possível realizar uma busca mais eficiente, dividindo o espaço de busca em áreas menores. Consequentemente, as partículas podem ser distribuídas em grupos, permitindo que explorem esses espaços simultaneamente. Estas estratégias permitem utilização de um número maior de partículas, o que possibilita a otimização na busca por uma melhor solução. Podem se beneficiar deste paralelismo, problemas que requeiram soluções complexas e que exijam poder computacional substancial e avaliações de aptidão de forma intensiva. A Seção 2.1 apresenta os modelos de paralelização caracterizando os diferentes tipos de mensagens existente entre processadores. A Seção 2.2 apresenta variações no algoritmo PSO, com o objetivo de paralelizar o PSO. A Seção 2.3 apresenta as topologias de troca de mensagens que utilizaremos em nossos experimentos. A Seção 2.4 apresenta as considerações finais.

2.1 Modelos de paralelização

Nos modelos paralelos quando fazemos uso de diversos processadores, a troca de informações entre os componentes do arranjo, seguem regras específicas as quais ficaram conhecidas como modelos de paralelização. Estes modelos seguem as mesmas características e arranjos apresentados no Capítulo 1, quando abordamos as topologias como Malha 2D, Mestre-trabalhador e Árvore.

Como as denominações para os modelos variam conforme os autores, utilizaremos neste trabalho o modelo de paralelização global e o modelo de múltiplas populações também conhecido como de granularidade grossa. Segundo (FOSTER, 1995) no modelo conhecido como de granularidade fina, quando dividimos a tarefa principal em outras menores, estas realizam um grande volume de troca de dados durante a etapa de comunicação enquanto as conhecidas como de granularidade grossa realizam um volume baixo, pois a tarefa principal é executada quase em sua totalidade pelas múltiplas populações. No modelo de paralelização Global, na qual incluímos a estratégia Mestre-trabalhador, não ocorre comunicação entre os processadores trabalhadores. As avaliações do cálculo da função objetivo, são realizadas em paralelo nos processadores trabalhadores. Conforme (GÜLCÜ; KODAZ, 2015), o processador mestre fica responsável por enviar frações do enxame e os parâmetros necessário a execução do PSO, além de definir o melhor global que será enviado aos trabalhadores. Esta estratégia, além da em Anel e Malha 2D, foram escolhidas para serem experimentadas neste trabalho e voltarão a ser abordadas novamente a partir da seção 2.3.1, quando aprestaremos detalhes e algoritmos. Já no modelo de granularidade grossa, nas quais se incluem a topologia em Anel e Malha 2D, o enxame principal é dividido em subenxames que são distribuídos nas unidades processadoras chamadas neste trabalho de ilhas. Na topologia em anel, cada processador se comunica com os processadores da direita e esquerda respectivamente e na topologia Malha 2D cada processador se comunica com 2, 3 ou até 4 processadores dependendo de sua posição na malha.

2.2 Algoritmos Paralelos e o PSO

Em (WOLPERT; MACREADY, 2005), surgiu o teorema chamado *no free lunch*, ou seja, não existe almoço grátis. Este artigo afirma que quaisquer algoritmos de otimização são

equivalentes quando seus desempenhos são comparados com funções problemas ou em nosso caso específico com funções objetivos diferentes. Mesmo assim, soluções têm sido propostas com o objetivo de melhorar a performance destes algoritmos.

2.2.1 Algoritmos paralelos síncronos e assíncronos

Com o objetivo de melhorar a eficácia dos algoritmos que podem ser utilizados nos modelos de paralelização apresentados, surgiu em (SCHUTTE et al., 2004) uma primeira proposta de um algoritmo paralelo chamado de *Parallel Syncronous Particle Swarm Optimization-PSPSO*, dentro desta proposta surgiram as implementações paralela do PSO nas modalidades síncrona e assíncrona. O funcionamento do PSPSO, fica melhor entendido na ilustração da Figura 7.

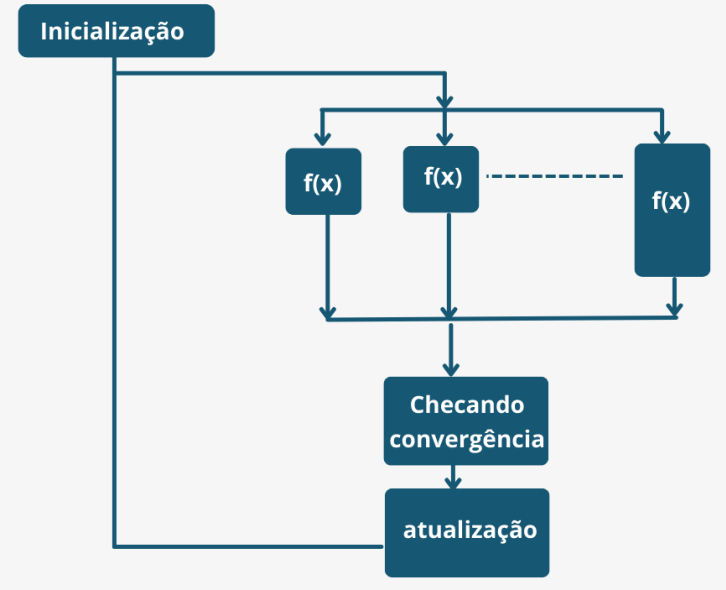


Figura 6: Diagrama de blocos PSPSO.Adaptado de (KOH et al., 2006)

Na modalidade síncrona, conforme utilizamos neste trabalho, todas as partículas são atualizadas dentro da mesma iteração, ou seja, o algoritmo só segue adiante após aguardar a atualização de todas as partículas do enxame. Esta modalidade segundo (VENTER; SOBIESZCZANSKI-SOBIESKI, 2006), pode gerar a ociosidade em algum processador até que todas os cálculos da função objetivo sejam efetuados para todas as partículas. Com o intuito de solucionar este problema surge a proposta de uma versão assíncrona chamada de *Parallel Asyncronous Particle Swarm Optimization-PASPSO*. Conforme ilustrado no diagramada de blocos na Figura 7, nenhuma partícula precisa aguardar a conclusão das demais. Neste modelo, nenhum processador fica ocioso durante o processo, pois o algo-

ritmo não irá interferir na ordem em que as partículas são avaliadas conforme sua função objetivo (KOH et al., 2006).

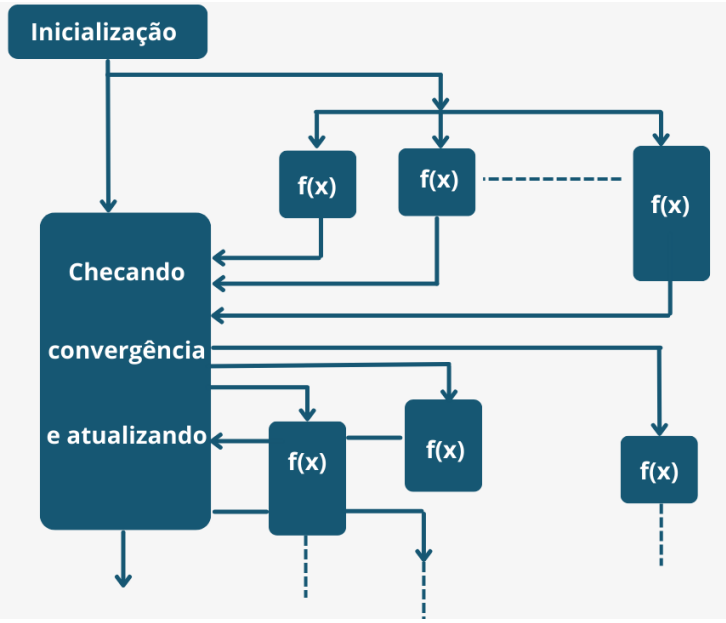


Figura 7: Diagrama de blocos PAPSO. Adaptado de (KOH et al., 2006)

2.2.2 Algoritmo cooperativo

O cooperativismo, que utilizaremos neste trabalho, foi introduzido por (BERGH; ENGELBRECHT, 2004). Neste modelo, o enxame original será dividido em subenxames contendo apenas uma parcela do enxame original. A cooperação se dará entre estes subenxames, que trocarão informações de formas diferentes conforme os modelos de paralelização que apresentamos neste capítulo. A troca de informações tem como objetivo trazer melhoria ao algoritmo original de forma a trazer maior celeridade e assertividade. Outro pilar importante neste modelo é a divisão do espaço de busca aonde procuraremos nossa solução, por isso cada subenxame ficará dedicado apenas ao seu espaço de busca. A Figura 10 ilustra o modelo proposto pelo PSO Cooperativo (Cooperative Particle Swarm Optimization - CPSO), com a composição do vetor final que trará a solução global. Em nossos experimentos dividimos o enxame inicial em vários grupos, aos quais chamamos de ilhas de subenxames.

2.2.3 Algoritmo paralelo PPSO

A ideia do algoritmo PPSO (*Parallel* PSO) é aproveitar a capacidade de processamento paralelo para acelerar o processo de otimização. Cada processador é responsável por atu-

alizar sua parcela da população de forma independente, e compartilhar suas informações entre os demais. Desse modo as atualizações referentes aos cálculos de função de aptidão, velocidade, deslocamento e *pbest*, ocorrem simultaneamente em diferentes partes do espaço de busca. Para que haja sincronia neste processo os novos cálculos para os valores de velocidade e posição só são efetuados após a definição do valor de *Gbest*. O fluxograma para este algoritmo é mostrado na Figura 8, quando utilizamos uma topologia em estrela em que a troca de mensagens é feita entre todas as partículas do enxame.

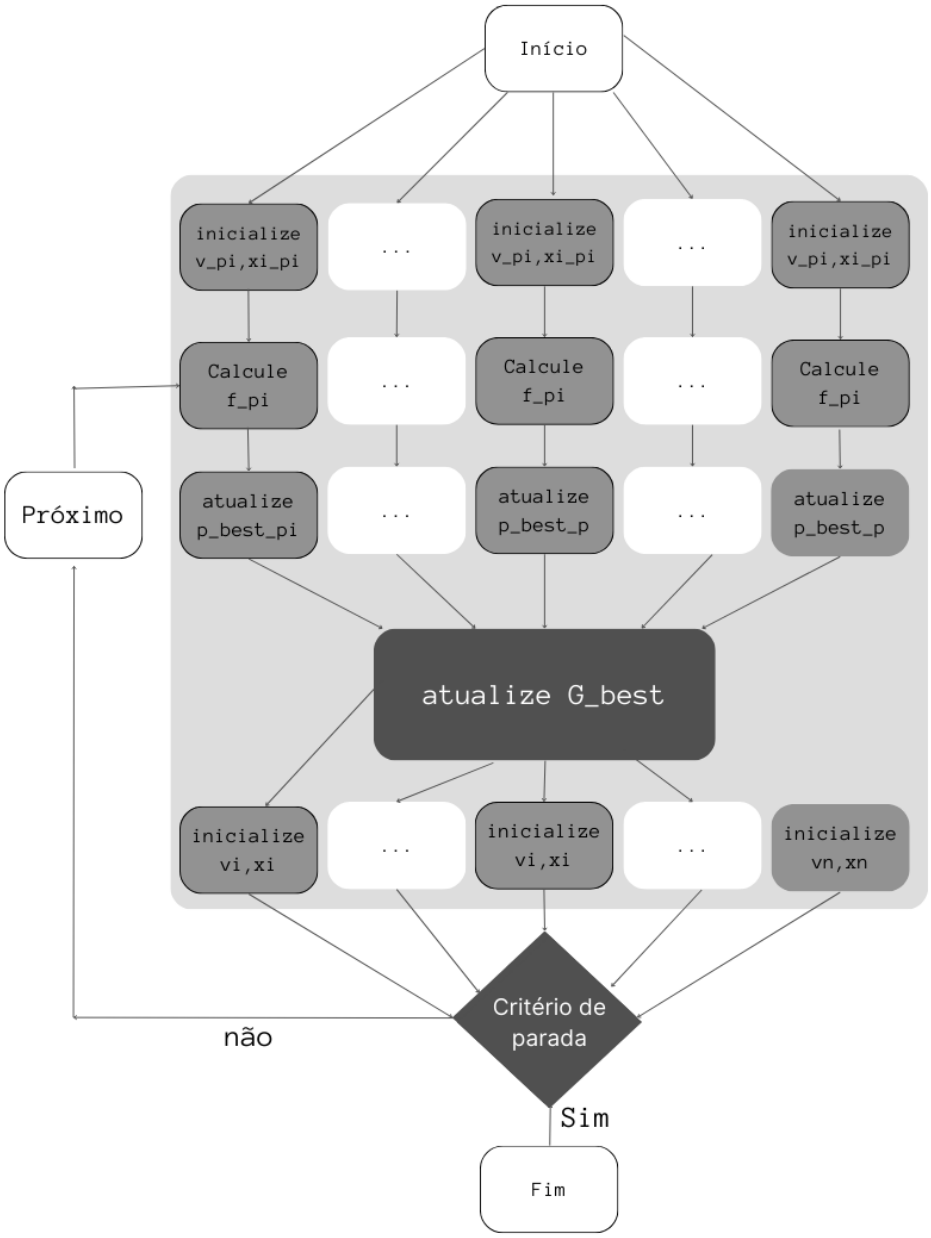


Figura 8: PPSO com topologia em estrela. Adaptado de (CALAZAN et al., 2013)

2.2.4 Algoritmo paralelo PDPSO

No algoritmo PDPSO (CALAZAN et al., 2013)(*ParallelDimensionPSO*), o objetivo é distribuir o paralelismo de forma mais refinada comumente chamado de granularidade fina. Este paralelismo pode ser conseguido, por exemplo, quando conseguimos realizar simultaneamente algum dos processos internos do cálculo como as dimensões associadas a cada partícula. Em um problema envolvendo um grande número de dimensões, todas poderiam ser executadas na mesma iteração ao invés de termos um número de iterações correspondente ao valor do número de dimensões. Na Figura 9 , podemos observar os blocos em profundidade onde são realizadas as operações paralelas conforme as dimensões.

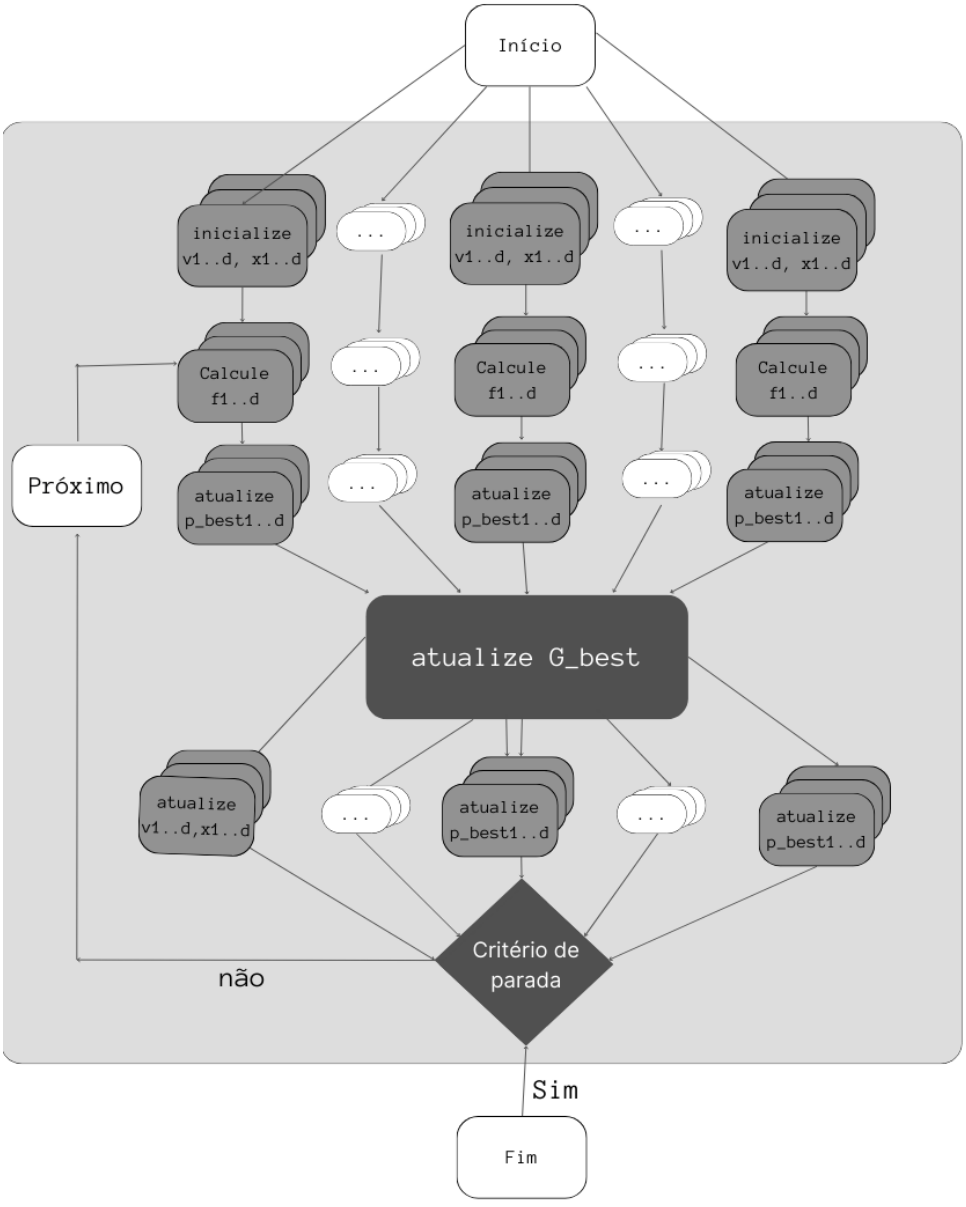


Figura 9: PDPSO com topologia em estrela. Adaptado de (CALAZAN et al., 2013)

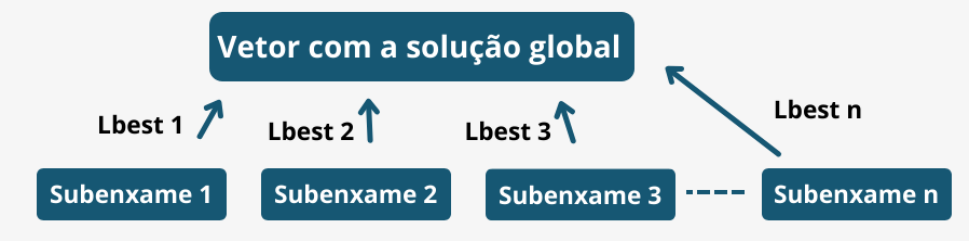


Figura 10: Vetor Solução Global (CPSO)

2.3 Algoritmo Cooperative PSO (CPSO) e topologias utilizadas

Nesta seção abordaremos a utilização do algoritmo CPSO, em três modelos dos paralelização apresentados neste capítulo. Na Seção 2.3.1 descrevemos sobre o modelo de paralelização Mestre-trabalhador. Na Seção 5.5.2 sobre o modelo em anel e finalmente na seção 2.3.3 sobre o modelo em malha 2D.

2.3.1 Topologia Mestre-trabalhador utilizando CPSO

Quando utilizamos a estratégia Mestre-trabalhador, ilustrada na Figura 11, fazemos uso de um processador principal chamado mestre. Os demais processadores, chamados trabalhadores, ficam responsáveis pela execução do algoritmo. A ideia é que o enxame principal seja dividido em enxames menores de forma que possam explorar de forma independente o espaço de busca. A divisão das partículas no espaço será feita proporcionalmente ao número de processadores, ou seja, quando consideramos o espaço [0 50], 2 PEs e 200 partículas, cada um dos PEs ficará dedicado respectivamente aos espaços [0 25] e [25 50]. Quanto ao número de partículas, cada um dos PEs receberá a informação com o número total do enxame e fará o cálculo para definir quantas partículas utilizará, fazendo a divisão entre o número de partículas total do enxame e de PEs. No exemplo citado acima para 2 PEs e 200 partículas, cada um trabalhará com 100 partículas. Inicialmente o processador mestre envia para todos os trabalhadores os parâmetros necessários a execução do algoritmo, como o número de partículas $npart$ e os coeficientes social e cognitivo, c_1 e c_2 respectivamente e coeficiente de inercia w. No Algoritmo 3, apresentamos as etapas efetuadas pelo processador mestre. O algoritmo definido para o processador trabalhador é mostrado no Algoritmo 4. Cada processador trabalhador recebe os parâmetros e inicialmente fazem o cálculo para o número de partículas do subenxame. Na sequência iniciam o cálculo para o PSO, conforme o Algoritmo 1, e prosseguem até a condição de

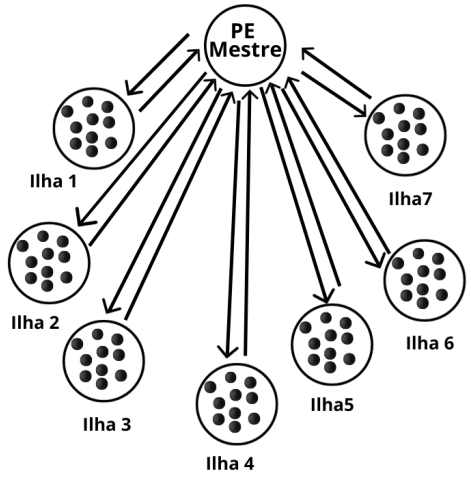


Figura 11: Estratégia de paralelização Mestre-trabalhador

parada, que em nossos experimentos é o valor de mínimo para a função ou o percentual de conversão das partículas em direção ao valor de mínimo. A cada iteração, cada processador trabalhador i envia seu valor de melhor local $lbest_i$, para o processador mestre, que compara este valor com os demais recebidos e decide qual o melhor resultado para a iteração. Este valor é armazenado na variável $lbest_m$ e, em seguida enviado para os processadores trabalhadores, que passam a ter ciência do melhor local encontrado dentre todos os processadores na iteração. Este processo continua até que seja satisfeita a condição de parada com o processador mestre apresentando o resultado final.

Algoritmo 3 Procedimento processador Mestre

1: **inicializar** c_1, c_2, $npart$, $nproc$;
2: **enviar para trabalhadores** c_1, c_2, $npart$;
3: **receber** $lbest_1$;
4: $lbest_m = lbest_i$;
5: **repita**
6: **para** $i := 1 \rightarrow nproc$ **faça**
7: **receber** $lbest_i$;
8: **se** $lbest_i < lbest_m$ **então**
9: $lbest_m = lbest_i$
10: **fim se**
11: **fim para**;
12: **até** Condição de parada;
13: **retorna** enviar para PEs trabalhadores melhor Lbest;

2.3.2 Topologia em Anel utilizando CPSO

O paralelismo no PSO pode ser alcançado por meio de várias técnicas de particionamento, envolvendo a divisão do enxame principal em partes menores, chamadas subenxames, que

Algoritmo 4 Procedimento processador trabalhador

1: **receber** c_1, c_2, $npart$;
2: **criar** Subenxame com n partículas $npart$;
3: **repita**
4: **calcular** PSO;
5: **até** Condição de parada;
6: **enviar** $lbest_i$ para o processador mestre;

podem ou não cooperar entre si. Na estratégia em anel (ENGELBRECHT, 2005), como mostrado na Figura 12, as partículas só têm acesso a informações sobre a melhor solução dos subenxames à sua direita e à sua esquerda. Nessa estratégia, soluções que poderiam potencialmente estar mais próximas do ótimo não são exploradas tão extensivamente. Este fato se dá devido termos uma troca limitada de informações entre as partículas o que pode causar uma convergência prematura para um mínimo local e não global (KENNEDY; MENDES, 2002). Os subenxames serão mapeados entre processadores vizinhos no nosso ambiente de simulação caracterizando assim a estratégia que está sendo utilizada. Utilizaremos para esta finalidade o mapeamento estático disponível no ambiente de simulação. Os processadores vizinhos trocarão informações para se beneficiarem dos melhores resultados referentes ao melhor local dos processadores localizados à direita e à esquerda, definidos respectivamente por dproc e eproc conforme ilustra o Algoritmo 5. Os parâmetros w , c_1 e c_2 correspondem respectivamente, conforme definimos no Capítulo 1 , ao fator de inércia, e os coeficientes de aprendizado social e cognitivo. O número de partículas n do subenxame é definido pelo número total de partículas $npart$ dividido pelo número de processadores utilizados $nproc$. As variáveis $lbest$ e $gbest$, representam respectivamente os valores de melhor local e melhor global. Após a inicialização, definição de parâmetros e número de partículas do subenxame, o cálculo do PSO é executado por cada processador que ao término da iteração terá seu valor de melhor solução local definido por $lbest$. Este valor é enviado aos processadores vizinhos que na etapa seguinte verifica para a iteração atual, qual o melhor valor de mínimo local após a troca de mensagens entre os processadores. Cada processador compara seu resultado $lbest$ com os valores recebidos da direita e esquerda respectivamente $lbest_d$ e $lbest_e$. A melhor solução é encontrada e o valor armazenado na variável $lbestn$. Estabelecemos, como condição de parada, o percentual de partículas, definido durante a execução das simulações, que convirjam para o valor de mínimo dentro do espaço de busca definido para cada processador. Satisfeita esta condição

de parada, cada processador indicará através da variável *lbestm* o melhor valor dentre as soluções encontradas a partir da vizinhança da direita e esquerda.

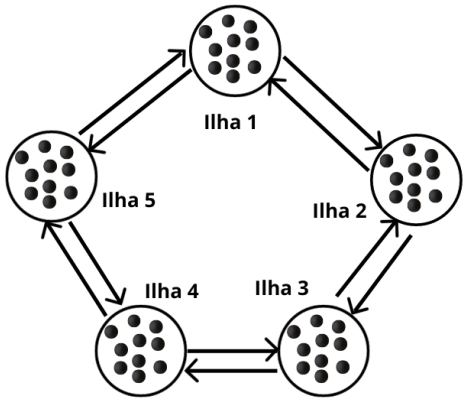


Figura 12: Estratégia de paralelização Anel

Algoritmo 5 PSO paralelo com estratégia em Anel

1: **inicializar** w ,c_1, c_2, $npart$, $nproc$, $pbest$;
2: **n =** $npart/nproc$;
3: **Criar** enxames com n partículas do total npart;
4: **repita**
5: **Calcular** PSO ;
6: **Enviar** para os processadores da direita e esquerda seu valor de melhor local $lbest$;
7: **Receber** dos processadores da direita e esquerda valores de melhor local $lbest_d$ *e* $lbest_e$;
8: **se** $lbest_d < lbest$ **então**
9: $lbestn = lbest_d$
10: **senão**
11: $lbestn = lbest$;
12: **fim se**
13: **se** $lbest_e < lbestn$ **então**
14: $lbestn = lbest_e$;
15: **fim se**
16: **até** Condição de parada;
17: **se** $lbest < lbestn$ **então**
18: $lbestm = lbest$;
19: **senão**
20: $lbestm = lbestn$;
21: **fim se**
22: **retorna** $lbest$

2.3.3 Topologia Malha 2D utilizando CPSO

Na estratégia de malha 2D mostrada na Figura 13, ocorre uma troca de informações entre, no máximo quatro subgrupos. Como resultado, o algoritmo pode coletar mais informações, abrangendo uma área maior com soluções potenciais para o problema. Em nosso

ambiente de simulação, isso ocorre quando os processadores vizinhos estão localizados à direita, esquerda, acima e abaixo de cada um dos processadores que fazem parte da topologia. Essa troca de mensagens mais ampla permite que cada processador se beneficie dos resultados encontrados pelos outros na topologia. Esse algoritmo, conforme apresentado em Algoritmo 6, é semelhante ao descrito para a estratégia em anel e difere apenas no número de processadores envolvidos na troca de mensagens. Inicialmente os parâmetros w (fator de inércia), c_1 (coeficiente de aprendizado social) e c_2 (coeficiente de aprendizado cognitivo) são inicializados. Na sequência é realizado o cálculo do número de partículas n do subenxame conforme o número de processadores utilizados $nproc$. Conforme utilizamos na topologia em Anel, as variáveis $lbest$ e $gbest$, representam respectivamente os valores de melhor local e melhor global. É realizado o cálculo para o PSO e em seguida o resultado $lbest$ é enviado para os processadores vizinhos. Na sequência, cada processador efetua a comparação com os valores recebidos dos vizinhos. O resultado de cada processador é então comparado com os valores dos vizinhos definidos respectivamente por lbest da direita ($lbest_d$), lbesta da esquerda ($lbest_e$), lbest cima ($lbest_c$) e lbest baixo ($lbest_b$). A melhor solução é encontrada e o valor armazenado na variável $lbestn$, que será atualizada a cada iteração com o melhor valor conhecido pelo processador, que poderá ser o valor que encontrou ou valores recebidos da vizinhança. Satisfeita esta condição de parada, cada processador armazenará na variável $lbestm$ o melhor valor final dentre as soluções encontradas que pode ser o valor que encontrou ou o melhor valor calculado a partir da comparação com os resultados finais encontrados pelos processadores da vizinhança.

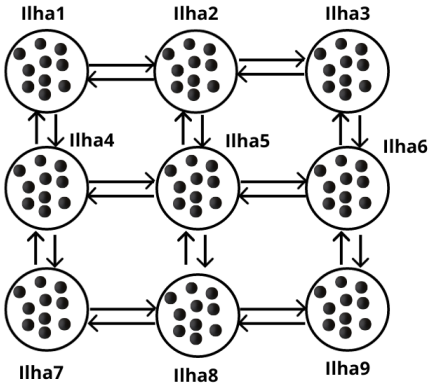


Figura 13: Estratégia de paralelização Malha 2D

Algoritmo 6 PSO paralelo com estratégia Malha 2D

1: **inicializar** w ,c_1, c_2, $npart$, $nproc$, $pbest$;
2: **Criar** subenxames com n partículas $npart/nproc$;
3: **repita**
4: **calcular PSO;**
5: **Enviar** para os processadores da direita, esquerda, cima e baixo seu valor de melhor local $lbest$;
6: **receber** valores $lbest$ dos vizinhos da direita, esquerda, cima e baixo: $lbest_d$, $lbest_e$, $lbest_c$ e $lbest_b$
7: **se** $Proc_d = Verdadeiro$ **então**
8: **se** $lbest_d < lbest$ **então**
9: $lbest = lbest_d$
10: **fim se**
11: **fim se**
12: **se** $Proc_e = Verdadeiro$ **então**
13: **se** $lbest_e < lbest$ **então**
14: $lbest = lbest_e$
15: **fim se**
16: **fim se**
17: **se** $Proc_c = Verdadeiro$ **então**
18: **se** $lbest_c < lbest$ **então**
19: $lbest = lbest_c$
20: **fim se**
21: **fim se**
22: **se** $Proc_b = Verdadeiro$ **então**
23: **se** $lbest_b < lbest$ **então**
24: $lbest = lbest_b$
25: **fim se**
26: **fim se**
27: **até** Condição de parada;
28: **se** $lbest < lbestn$ **então**
29: $lbestm = lbest$;
30: **senão**
31: $lbestm = lbestn$;
32: **fim se**
33: **retorna** $lbest$

2.4 Considerações Finais

Neste capítulo foram apresentadas estratégias de paralelização e porque utilizar. Foram descritos alguns dos modelos de paralelização que podemos encontrar, sendo dado foco aos modelos Mestre-trabalhador, Anel e Malha 2D, que serão objetos de nossos experimentos com o objetivo de medir o desempenho. Abordamos os algoritmos síncronos e assíncronos utilizando paralelismo. Em seguida, foi apresentado o conceito de ilhas de subenxames e o algoritmo cooperativo CPSO, que utilizaremos em nossos experimentos no Capítulo 5.

Capítulo 3

TRABALHOS RELACIONADOS

ESTE capítulo apresenta alguns trabalhos relacionados ao algoritmo PSO e suas formas paralelas. Muitas são as vantagens que podem ser obtidas com a paralelização como redução do tempo de execução, aumento da capacidade de busca global e escalabilidade. Os trabalhos expostos a seguir apresentam as diferentes propostas para a paralelização do PSO bem como o *hardware* e *softwares* utilizados. Neste capítulo, serão apresentadas algumas soluções paralelas e os resultados obtidos, bem como um comparativo entre os diferentes tipos de estratégias. Tais propostas foram categorizados em três seções: na Seção 3.1 são discutidos os trabalhos que utilizam as estratégias de paralelização baseadas em Unidade Central de Processamento (Central Processing Unit - CPU); a Seção 3.2 apresenta trabalhos focados na utilização de Unidades Gráficas de Processamento (Graphics Processing Units - GPU). A Seção 3.3 apresenta trabalhos que implementam o paralelismo utilizando arquiteturas de *hardware* dedicadas. A Seção 3.4 apresenta as considerações finais.

3.1 Estratégias de paralelização baseadas em CPU

As estratégias que utilizam CPU dividem um problema maior em outros menores denominados subtarefas, que podem ter sua execução otimizada a partir da utilização dos vários núcleos de processamento na CPU. Conforme (LALWANI et al., 2019) estas estratégias, tiram proveito dos vários núcleos existentes em uma única CPU, que não necessariamente precisam ser físicos mais também virtuais além de fazer uso uma ou mais CPUs. Muitas são as estratégias que podem ser utilizadas com este objetivo. Podemos citar, como exemplo, a utilização de bibliotecas presentes em vários *softwares* como: Matlab Tool Box, R Parallel package, Julia: Parallel MapReduce e Parallel Computing module in Python.

Em (VENTER; SOBIESZCZANSKI-SOBIESKI, 2002), é mencionado que um PSO paralelo pode ser implementado simplesmente adicionando o cálculo da função objetivo em seu algoritmo, que será realizado em paralelo por outros processadores sem alterar a lógica do algoritmo. Nesse modelo síncrono, há uma limitação de espera para o processamento em todos os escravos antes que o cálculo possa prosseguir, o que leva a uma eficiência reduzida. Neste formato um grande número de processadores podem ficar ociosos, reduzindo assim, a eficiência quando utilizamos algoritmos síncronos.

Já em (KOH et al., 2006), a proposta é evitar a necessidade de sincronização no final de cada iteração, prevenindo a interrupção no cálculo da função objetivo e, assim, aprimorando a eficiência. Em termos de implementação, PSO paralelo síncrono e assíncrono diferem apenas na etapa de otimização. Como não há ponto de sincronização, apenas uma ordem global de cálculo é necessária antes do início do procedimento de otimização, enquanto nos algoritmos síncronos, a sincronização ocorre no início de cada iteração.

O OpenMP *(Open Multi-Processing)* (CHAPMAN; JOST; PAS, 2008) é uma API que oferece suporte à programação paralela em linguagens como C, C++, e Fortran. A principal finalidade do OpenMP é facilitar o desenvolvimento de software que pode ser executado em sistemas com múltiplos processadores ou núcleos de CPU que fazem acesso a uma região comum de memória. Um recurso presente no OpenMP é a sincronização entre as tarefas de maneira automática facilitando sua utilização. O Modelo de Programação é baseado em Threads (SULZBACH et al., 2014) o que possibilita a execução de tarefas que podem ser divididas em partes independentes, cada uma sendo executada por uma thread separada. Em (CALAZAN et al., 2013), é discutida uma solução que utiliza uma arquitetura baseada em OpenMP. Os testes foram realizados em uma *Workstatios* SGI Octane III, que possui 16 núcleos de processamento. Foram efetuados testes e comparativos entre diferentes estratégias de paralelização para o PSO.

No artigo de (ALTINOZ et al., 2014), é introduzida a técnica de computação distribuída usando várias máquinas por meio da Interface de Troca de Mensagens (*Message Passing Interface* - MPI). A tarefa principal é dividida em tarefas menores que são distribuídas entre várias CPUs. Cada tarefa pode compartilhar os dados obtidos com as demais tarefas fazendo uso de troca de mensagens. No caso do PSO, estas mensagens podem ser utilizadas para troca de informações entre as partículas do enxame. A utilização da MPI

permite que uma aplicação seja executada em diferentes plataformas desde que suportem a MPI, sem a necessidade de alterações no código.

3.2 Estratégias de paralelização baseadas em GPU

Uma GPU (Graphics Processing Unit) é um hardware composto por uma matriz aonde são dispostos multiprocessadores denominados Streaming Multiprocessors (SMs) (KIRK; WEN-MEI, 2016). Cada GPU contém vários SMs, que são unidades de processamento individuais responsáveis por executar tarefas em paralelo. Com a utilização da computação utilizando GPU é possível executar tarefas intensivas em computação paralela. Para facilitar a programação em paralelo utilizando GPU, a NVIDIA criou a arquitetura CUDA permitindo que desenvolvedores possam escrever código CUDA utilizando extensões da linguagem de programação C na execução de funções que serão executadas em paralelo nas GPUs. Em (KUMAR; SINGH; PAUL, 2013) para otimizar o cálculo no PSO, o enxame inicial de partículas foi dividido em subenxames que agora passam a ser representados por vetores que evoluem de forma individual e simultânea. A execução deste paralelismo foi implementada utilizando CUDA juntamente com linguagem C.

Em (CALAZAN et al., 2013) a proposta é a utilização GPU com CUDA na implementação paralela do PSO utilizando o algoritmo PDPSO (Parallel Dimension PSO), aonde cada dimensão do problema é executada em paralelo. Nesta proposta, cada subenxame é implementado como um bloco de threads e cada dimensão é mapeada para uma *thread* distinta e executada de forma paralela. A alocação das tarefas computacionais ocorre em um nível mais refinado de granularidade, pois a atualização da velocidade e posição das partículas são realizados simultaneamente.

O padrão de programação para computação paralela OpenACC (DIETRICH; JUCKELAND; WOLFE, 2015), foi criado de forma a permitir o uso de diretivas em linguagens como C, C++ e Fortran. Segundo (FARBER, 2016), a utilização destas diretivas é que permitem expressar o paralelismo além de oferecer alto desempenho e portabilidade entre vários tipos de arquiteturas, incluindo CPUs e GPUs. A utilização da CUDA, por outro lado, requer em função dos diferentes tipos de memórias encontrados nas GPUs, que o desenvolvedor configure seus acessos a memória global e a cache além de configurar a quantidade e disposição das threads de forma que possa atingir o melhor desempenho, o que pode gerar uma reescrita mais substancial do código (BURIOL; ARGENTA, 2009).

Em (TOLEDO et al., 2022) foi realizado um estudo sobre o impacto ao utilizarmos uma combinação de OpenACC com CUDA Graph na execução do PSO em sua versão original apresentada por (SHI et al., 2001). Foi utilizado um enxame com 1000 partículas e realizadas simulações durante 10.000 iterações. Essa abordagem foi capaz de alcançar uma redução importante no tempo de execução com um speedup variando de 2 a 4.

Em (HUNG, 2012), a proposta consiste na utilização de um modelo utilizando várias GPUs e implementando o que foi chamado de GPU - *accelerated* PSO (GPSO). Ocorreu uma grande redução no tempo computacional, alcançando alta eficiência em paralelo ao utilizar um grande número de partículas. No experimento foi resolvido um problema contendo 100 dimensões e com 65.536 partículas. O GPSO alcançou acelerações de até 280 e 83 em uma GPU NVIDIA Tesla C1060 de 1,30 GHz em relação a uma CPU Intel Xeon-X5450 de 3,00 GHz executando no modo single-core e quad-core respectivamente. O ponto-chave para alcançar eficiência foi aproveitar o poder dos multiprocessadores de vários núcleos e otimizar a largura de banda da memória na GPU.

3.3 Estratégias de paralelização baseadas em hardware

Principalmente com a necessidade do uso em aplicações embarcadas, surgiu a abordagem de co-design de *hardware* e *software* utilizando sistemas que podem ser programados em um chip (LI et al., 2010). A estrutura modular passou a ser bastante utilizada para aprimorar o despenho do PSO. Módulos específicos para atualizar velocidade e posição de partículas, e para cálculo da função objetivo passaram a ser implementados em processadores denominados *softcore*, que permitem ser customizados e sintetizados em uma FPGA*(Field Programmable Gate Array)*. As FPGAs permitem a criação de *hardware* especializado para paralelização de cálculos utilizando *pipeline*, permitindo ampla paralelização e customização para acelerar o desempenho do algoritmo. (XU et al., 2022).

Em (CALAZAN et al., 2013) é discutida uma solução que utiliza uma arquitetura baseada em hardware e software, empregando co-design para implementar o algoritmo PSO. O módulo para atualização da velocidade e posição da partícula é implementado em *hardware*, enquanto o módulo para a função objetivo é implementado em *software*. O *hardware* utilizado foi uma FPGA (*Field Programmable Gate Arrays)*, que devido ao aumento significativo da área ao incrementar o número de partículas, acabou limitando a implementação com apenas 10 partículas. A conclusão neste trabalho é que mesmo

com esta limitação a solução ainda é vantajosa, tendo em vista o ganho em desempenho alcançado, apesar do aumento da área requerida.

Em (TEWOLDE; HANNA; HASKELL, 2012), a proposta é, além de melhorar a eficiência da execução, apresentar um projeto modular, flexível e reutilizável para resolver diferentes problemas de otimização. O objetivo é acelerar ainda mais a execução do algoritmo usando módulos de hardware escaláveis, sendo uma das principais aplicações o seu uso em pequenos sistemas embarcados. Neste modelo, o enxame principal é dividido em 2, 3 ou 5 subenxames, que operam simultaneamente. Para os testes foram utilizados dois processadores já utilizados normalmente para aplicações embarcadas: o microcontrolador Freescale MC9S12DP256B e o núcleo de processador Xilinx MicroBlaze. Os valores de *speedup* varianram entre 2,09 e 3,89 quanto utilizado a Função Esfera e entre 2,87 e 6,96 quando utilizado a função Rosenbrock.

Em (COSTA et al., 2019) é apresentada uma arquitetura utilizando FPGA, que realiza paralelamente a movimentação das partículas do enxame para, então, seguir com as operações em comum. Este modelo também permite a exploração de uma área mais diversificada dentro do espaço de busca. No experimento são utilizados quatro blocos dedicados denominados: partículas, comparação, Gbest e aptidão. O bloco partículas é responsável por calcular a nova posição de cada partícula e seu valor de aptidão. O bloco comparação ficou responsável por verificar entre dois valores de aptidão qual o menor. O módulo Gbest guarda os valores de coordenadas da melhor partícula. O módulo aptidão armazena o valor de aptidão correspondente as coordenadas armazenadas no módulo Gbest. O FPGA utilizado nos testes foi o Virtex 6 que possui 150.720 células (LUT) que podem ser utilizadas para implementar funções lógicas. Os resultados foram relevantes tendo sido encontrado *speedup* de até 212. Outro ponto de destaque no trabalho é a melhora na ocupação no *hardware* da FPGA.

Em (LI et al., 2010) é apresentada uma abordagem utilizando *hardware* e *software* fazendo uso de co-design. Esta proposta surgiu devido a necessária reformulação do *hardware* no caso de resolução de diferentes problemas e consequentemente de diferentes funções de avaliação. Foi utilizado um bloco em FPGA onde o módulo denominado acelerador ficou responsável por atualizar a posição e velocidade das partículas. O módulo de avaliação de aptidão compreende um módulo de *software* e outro de *hardware* para avaliar funções objetivo de forma a lidar com diferentes aplicações. Esses módulos foram

implementados em uma CPU Nios II e em uma FPGA, respectivamente. Esta abordagem permitiu uma maior flexibilidade quanto ao uso em diferentes aplicações ou ajuste de parâmetros sem a necessidade de um novo *hardware*. Conforme os testes executados, o bloco acelerador apresentou um *speedup* de 20 quando comparado com a versão em *software*. Foi constatado que para o cálculo da aptidão foi necessário alto poder computacional, sendo utilizado um processador de alto desempenho para que a arquitetura mostre todo o seu potencial.

Na proposta de *hardware* apresentada em (DAMAJ et al., 2020), a implementação da etapa de cálculo para a função objetivo, foi direcionada para dois sistemas. O primeiro via implementações do hardware em FPGA Cyclone II da Altera e o segundo para um computador multi-core de alta qualidade para implementações do *software*. Uma das ferramentas utilizadas para síntese e análise de *hardware* foi a Quartus da Intel que disponibiliza ferramentas de design para implementação em Field-Programmable Gate Arrays (FPGAs) e dispositivos System-on-Chip (SoC). Outra ferramente utilizada foi a ModelSim disponibilizada pela Siemens e que permite o uso de linguagens de descrição de hardware como VHDL, Verilog e SystemC. Os tempos apresentaram uma melhoria de execução significativos. A função de avaliação elíptica atingiu um *speedup* de 23300, e de 1777 com o uso da função Schwefels. As implementações de *hardware* foram realizadas em VHDL, contando com módulos dedicados para cálculo da aptidão, de pbest, gbest, velocidade e posição.

Em (XU et al., 2022) é apresentada a implementação de um controle preditivo não linear (NMPC) baseado em otimização por enxame de partículas (PSO). A plataforma utilizada foi a FPGA Altera Stratix III aonde foram implementados 2 módulos dedicados ao PSO: um para cálculo da função de aptidão e outro para cálculo do PSO com atualização de velocidade e posição. Nos comparativos finais entre as versões seriais e paralela, considerou-se para na versão serial a execução em um PC com processador Intel Core i5, operando a 3.4 GHz com código MATLAB. Outro cenário foi utilizando uma plataforma ARM com um processador Cortex-A9, operando a 800 MHz com código em C. Durante a execução em paralelo o tempo foi reduzido significativamente, porém com os recursos do hardware quase se esgotando, concluindo-se, assim, que esta é a etapa que consome mais recursos. Utilizando o FPGA para cálculo da aptidão e a atualização da velocidade e posição das partículas, foi atingido um *speedup* de 143 quando comparado com uma

solução utilizando o processador genérico I5 e *speedup* de 108 quando comparado com o processador ARM.

Em (KUI-TING; CHEN, 2014) é realizado um estudo sobre a utilização da solução empregando FPGA para síntese de módulos dedicados a execução do PSO. Dentre estes módulos podemos citar: o de geração de partículas, atualização da velocidade e posição, cálculo da função aptidão, escolha do Global best, e módulos específicos para armazenamento de valores de Global best e sua coordenada. Neste tipo de arquitetura, com muitos blocos, ocorre a necessidade de comunicação constante, gerando grande gasto de tempo. A proposta foi combinar vários módulos em apenas dois de forma a diminuir o tempo de comunicação. Estes dois módulos são o de cálculo de partículas, responsável por atualizar a velocidade e deslocamento, e o módulo de cálculo da função de aptidão. Para implementação foi utilizado uma arquitetura em FPGA Cyclone II DE2-70. O módulo de cálculo de partículas, que até então ficava preso em um algoritmo PSO específico, passou a ser programável permitindo alterações no algoritmo e seu sistema numérico sem necessidade de mudança no *hardware*. Utilizando esta arquitetura verificou-se um *speedup* de 887 quando comparado com a versão em software. Este trabalho também apresenta uma abordagem em co-design com a utilização conjunta de *hardware* e *software* de forma a dar maior flexibilidade. O diferencial proposto foi a inclusão, no bloco de cálculo da aptidão, de um processador programável de forma a possibilitar a utilização do PSO em diferentes aplicações. Nesta arquitetura foi possível obter um *speedup* de cerca de 3,6 a 12,8 em comparação com a solução em co-design.

Em (ARBOLEDA et al., 2009) é apresentada uma arquitetura totalmente desenvolvida em *hardware* utilizando FPGA e a linguagem de descrição VHDL. A arquitetura é formada por cinco componentes principais: unidade de enxame, unidade de avaliação, unidade de detecção de melhor individual, unidade de detecção de melhor global e um outro módulo chamado de máquina de estados finitos (FSM). A unidade de enxame atualiza velocidade e posição das partículas e a unidade de avaliação efetua o cálculo paralelo para a função objetivo. Os cálculos para identificação da melhor posição local e melhor posição global das partículas são efetuados respectivamente pelas unidade de melhor individual e melhor global. A função do módulo FSM é a de sincronizar todos os módulos. Foi utilizado para síntese o FPGA Xilinx Virtex e os testes apresentaram uma melhor

precisão quando comparado com a versão em software. O *speedup* alcançado foi de até 129 conforme a função utilizada.

Em (FARMAHINI-FARAHANI et al., 2010) é discutido uma proposta utilizando uma arquitetura que permite gerar um sistema completo utilizando apenas um chip conhecido como SOPC (*completesystem-on-a-programmable-chip*). A proposta é de uma arquitetura composta por um conjunto de processadores incorporados paralelos conectados por uma arquitetura de barramento on-chip. Existe assim a possibilidade de se projetar um *framework* aumentando o número de processadores conforme a necessidade. Nos testes realizados, com o objetivo de obter um melhor desempenho, foi implementado uma estrutura com multiprocessadores mestre-escravo em um único chip. Os resultados apresentaram um *speedup* de até 98.

3.4 Considerações finais

Todos os trabalhos citados denotam a importância de se explorar paralelismo tendo em vista a resolução de problemas complexos que consomem grandes quantidades de recursos computacionais e tempo de execução. Esta necessidade ocorre quando trabalhamos com uma grande quantidade de dados onde a paralelização pode ajudar a distribuir a carga de trabalho entre os diversos núcleos de processamento. Aplicações do dia a dia, em especial sistemas embarcados, e que necessitem de execução em tempo real, também se beneficiam com este melhor desempenho quando utilizamos paralelismo. Cada uma das estratégias utilizadas, apresenta sua usabilidade nos ambientes e propósitos a que se destinam. Na Seção 3.1 quando utilizamos apenas a própria CPU, técnicas de programação são empregadas visando o paralelismo, e linguagens de programação, como Python, podem ser utilizadas com esta finalidade. Também é possível o paralelismo utilizando alguns arranjos específicos como conexão entre máquinas através de troca de mensagens, com MPI, ou memória compartilhada, com OpenMP. Na Seção 3.2 a ideia é utilizar o poder de processamento das GPUs por serem componentes que, de forma nativa, já foram concebidos para realizar computação paralela com o objetivo de acelerar operações. Vários modelos de programação surgiram de forma a tirar proveito do poder das GPUs como uso de diretivas em C e CUDA, que traz uma API específica destinada a computação paralela. Com a necessidade cada vez maior de poder computacional, conforme descrito na Seção 3.2, surgiram as abordagens utilizando *hardware* ou um formato híbrido de *hardware* e

software utilizando co-design. Foram diversas as técnicas que trouxeram significativos aumentos de desempenho que podem ser comprovados com os *speedups* obtidos. Nas estratégias utilizando *hardware*, os vários blocos dedicados a execução de funções específicas no PSO puderam ser projetados e trazidos ao uso diário com a utilização de técnicas de síntese de circuitos utilizando FPGA.

Capítulo 4

AMBIENTE DE SIMULAÇÃO

NESTE capítulo, apresentamos o ambiente de simulação utilizado para a experimentação. Trata-se de um Sistema Embutido Multiprocessado *(Multiprocessor System on Chip - MPSoC)*, utilizando uma Rede Intrachip *(Network on Chip - NoC)* como canal de comunicação entre os processadores. A coordenação do fluxo de comunicação é feita a partir da utilização de um algoritmo de roteamento conforme os diferentes tipos de topologias. A Seção 4.1 apresenta os conceitos de Noc, algoritmos de roteamento e topologias de comunicação. A Seção 4.2 apresenta o roteador HERMES. A Seção 4.3 apresenta detalhes da plataforma de experimentação MEMPHIS. A Seção 4.4 apresenta as considerações finais.

4.1 Conceitos básicos

Um MPSoC é constituído por um conjunto de Elementos Processadores (PE) capazes de executar tarefas em paralelo. A utilização de um canal compartilhado ou barramento como topologia de comunicação entre os Elementos Processadores (PE) não é viável, uma vez que não oferece escalabilidade. A alternativa é empregar o conceito de Rede Intrachip que fazem uso de roteadores (BENINI; MICHELI, 2002). Estes gerenciadores de comunicação, que são os roteadores, fazem uso de pacotes para coordenar as trocas de mensagens em conjunto com algoritmos de roteamento. Este gerenciamento proporciona uma melhor utilização dos recursos, além de melhorar a escalabilidade e a eficiência energética. O uso da NoC, segundo (WOLF; JERRAYA; MARTIN, 2008), permitiu a melhor implementação de sistemas embutidos chamados de (*System-on-Chip*-SoC) facilitando a popularização de diversos dispositivos como celulares e tablets. A Figura 14 ilustra uma NoC genérica com vários processadores conectados. O nó é um componente fundamental, que serve como

ponto de conexão para os diferentes blocos que compõem o sistema. Funcionam como chaves sendo responsáveis pelo fluxo de dados entre processadores e memórias presentes na NoC, cuidando para que cada dado seja entregue no local correto.

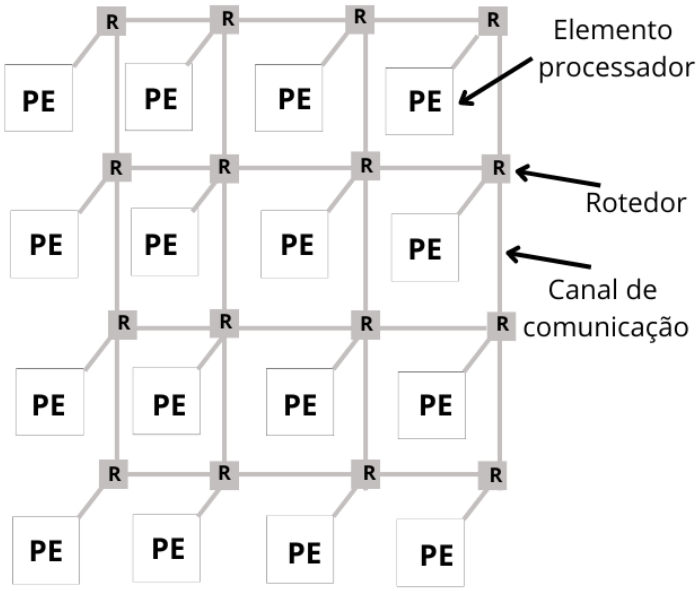


Figura 14: Arquitetura básica de uma NoC

De forma a melhor apresentar os termos citados, na Seção 4.1.1 apresentaremos os algoritmos de roteamento e na Seção 4.1.2 falaremos sobre as topologias de comunicação utilizadas.

4.1.1 Algoritmos de roteamento

Segundo (PASRICHA; DUTT, 2010) o algoritmo de roteamento é o responsável por fazer a entrega dos pacotes de dados ao destino de forma eficiente no menor tempo possível. Devem oferecer alternativas que possam surgir devido a possíveis falhas, criando caminhos alternativos. Também são importantes para evitar o aparecimento de situações conhecidas como *livelock* e *deadlock* onde dois ou mais processos ficam travados esperando que o outro tome uma ação. No *livelock*, os processos ficam ativos, mas acabam presos em um ciclo infinito de interação, enquanto no *deadlock* ficam permanentemente bloqueados conforme (ASHCROFT, 1975).

Uma das classificações para os algoritmos de roteamento é a forma como realizam o mesmo, que pode ser estático ou dinâmico. Na estática a configuração das rotas é feita de forma manual enquanto na dinâmica ocorrem trocas de informações entre os roteadores

de forma a determinar a melhor rota. Como exemplos de algoritmos dinâmicos temos o RIP (*Routing Information Protocol*) e o OSPF (*Open Shortest Path First*)

Outra tipificação possível é quanto a algoritmos que se baseiam em informações de distância que são recebidas de roteadores vizinhos como Bellman-Ford (CORMEN et al., 2001) e *Distance Vector Routing Protocol* (LU et al., 2003).

O algoritmo abordado em (GHEIN; FUNDAMENTALS, 2006) é o MPLS (*Multiprotocol Label Switching*), que não necessariamente foca na distância mais curta, mas também na disponibilidade de banda passante destes caminhos, ou seja, nem sempre a menor distância terá prioridade.

O algoritmo XY, muito popular, é utilizado pelo roteador Hermes que faz parte da nossa plataforma de experimentação. É utilizado especialmente em redes de malha bidimensional presentes em redes intra chip. Este algoritmo, para a entrega dos pacotes, faz a movimentação primeiramente no eixo X até localizar a coluna de destino na malha e em seguida faz a movimentação no eixo Y até o destino. Já o *Negative-first* (KARINIEMI; NURMI, 2004) que é bem parecido com o XY prioriza o movimento na direção oposta ao destino. Esta característica é uma tentativa de evitar o encontro com outros pacotes que estejam trefegando na rede. Em seguida se direciona ao destino para entrega do pacote.

4.1.2 Topologias de comunicação

As topologias de comunicação em uma NoC *(Network-on-Chip)*, descrevem como os nós de interconexão estão conectados, o que caracteriza a forma como irão realizar o roteamento de mensagens e recursos a que tenham disponibilidade. As topologias utilizadas, podem ser as mais diversas conforme os requisitos de cada projeto. Entre as topologias de comunicação mais comumente encontradas em uma NoC estão a Mestre-trabalhador, Anel e Malha 2D conforme abordado no Capítulo 2.

4.2 Arquitetura roteador HERMES

As redes intra chip, como alternativa aos barramentos, fazem uso de roteadores para gerenciar a comunicação, sendo esta uma parte crucial para o funcionamento eficiente dessa arquitetura. Esse controle de fluxo de dados entre os roteadores se dá através de mensagens estruturadas que fazem uso de pacotes gerenciados pelo roteador de comutação de pacotes (*Packet Switching-PS*). Para nossos experimentos estamos fazendo uso de um

roteador baseado na arquitetura HERMES (CARARA; CALAZANS; MORAES, 2008), estando seu diagrama básico ilustrado na Figura 15.

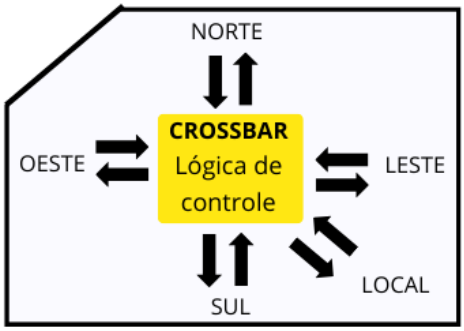


Figura 15: Arquitetura básica do roteador HERMES

A troca de mensagens com os roteadores vizinhos se dará através das portas existentes na NoC que é responsável por coordenar o fluxo de dados. Esta troca ocorre através de 4 portas de comunicações Leste, Oeste, Norte e Sul, todas bidirecionais. Também existe uma porta local que é responsável por alocar recursos como memória ou periféricos. Para melhor ilustrar como ocorre este gerenciamento podemos observar na Figura 16 a estrutura da mensagem que circula na NoC. Inicialmente esta é uma mensagem gerada por um elemento processador (*Processing Element*-PE) em forma de pacotes que são divididos em unidades de controle de fluxo (*Flow Control Unit*-flit). Esses *flits* são responsáveis por coordenar a sincronização entre os roteadores (PASRICHA; DUTT, 2010). Outro conceito importante é o de de *phit*, que define o tamanho da menor unidade de dados utilizado na NoC.

4.3 Plataforma MEMPHIS

Os experimentos realizados neste trabalho utilizaram a plataforma de simulação MEMPHIS (*Many-core Modeling Platform for Heterogeneous* SoCs). A MEMPHIS é uma plataforma de código aberto idealizada pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) através do grupo Grupo de Apoio ao Projeto de Hardware (GAPH) (RUARO et al., 2019). Este *framework* permite a criação de sistemas baseados em processamentos múltiplos utilizando um único chip (*Multiprocessor System on Chip*-MPSoC), viabilizando a interligação entre processadores, roteadores e periféricos. A síntese do *hardware* em FPGA (*Field Programmable Gate Array*) também é possível com a utilização do VHDL (*Very high speed integrated circuit Hardware Description Language*) e SystemC diretamente na

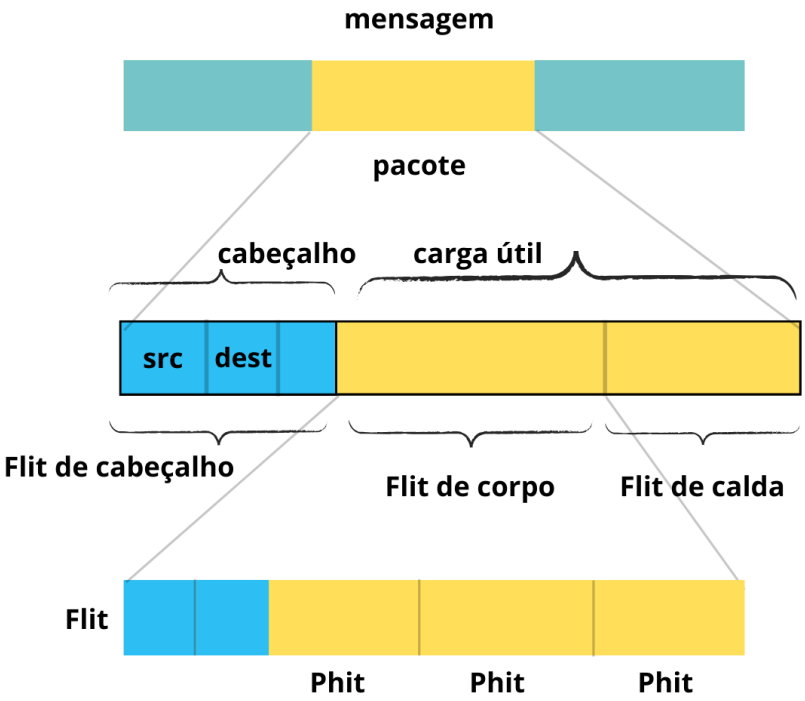


Figura 16: Mensagem, pacote e flits

plataforma. Como a MEMPHIS é uma plataforma acadêmica de pesquisa e está em constante desenvolvimento, apresenta algumas limitação que impactaram em nosso trabalho como falta de suporte a funções trigonométricas e operações de ponto flutuante.

Apresentaremos nesta seção informações sobre característica da arquitetura, forma de distribuição das tarefas entre os processadores e duas diretivas importantes que oferecem suporte à troca de informações entre os processadores. Dividimos esta seção da seguinte forma: A Seção 4.3.1 fala sobre a arquitetura da plataforma, na Seção 4.3.2 abordamos como ocorre a distribuição das tarefas entre os PEs , na Seção 4.3.3 abordamos como é realizada a troca de mensagens entre os PEs e finalmente na Seção 4.3.4 apresentamos a ferramenta debugger de onde obteremos nossos dados.

4.3.1 Características da arquitetura

A arquitetura MEMPHIS é composta por um modelo de muitos núcleos que podem ser heterogêneos ou não de forma a formar uma SoCs. O diagrama básico está ilustrado na Figura 17. Os elementos processadores (Processing Elements-PEs), são interconectados em uma malha 2D que por definição chamamos de núcleos de processamento de propósito geral (*General Purpose Processing Cores*-GPPC). Esta arquitetura suporta a integração dos PEs com os roteadores e periféricos.

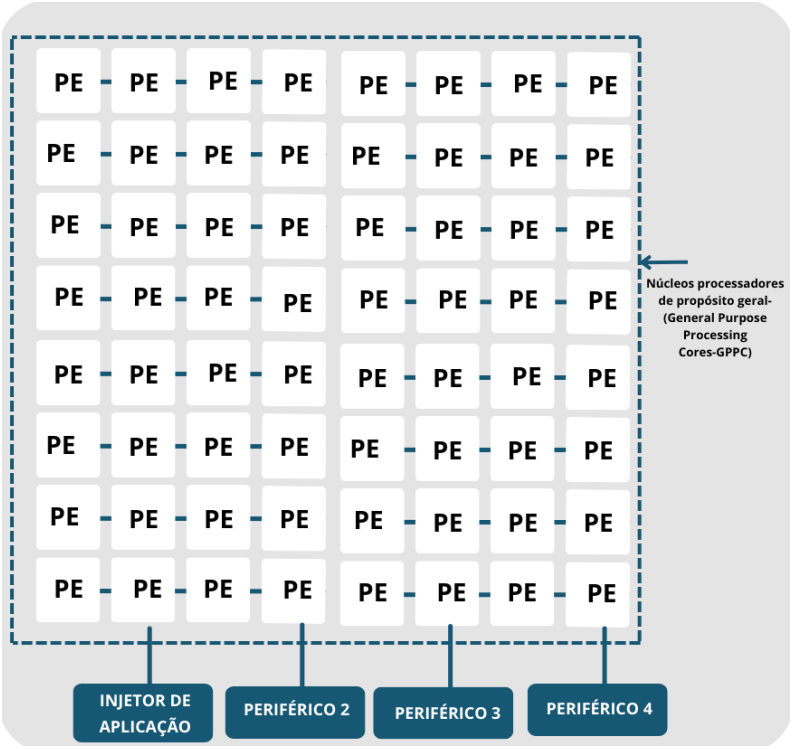


Figura 17: Arquitetura básica da MEMPHIS

O bloco com o elemento processador pode ser visto com mais detalhes na Figura 18. Cada PE é composto por uma Unidade Central de Processamento (*Central Processing Unit - CPU*), memória dinâmica local (*Dynamic Random Access Memory - DRAM*) e interface de memória e uma interface de acesso direto a memória (*Direct Memory Network Interface*- DMNI). Em nossos experimentos utilizamos o PE padrão disponibilizado pela MEMPHIS que utiliza como CPU o Plasma v2. Este processador possui 32 bits e suporte a 3 estágios de *pipeline* que possibilitam o paralelismo na execução de instruções. O acesso a memória pode ser feito simultaneamente pelo processador ou via DMNI fazendo uso da interface de porta dupla.

4.3.2 Distribuição de tarefas

A arquitetura MEMPHIS permite a adição de periféricos como memórias, aceleradores para processamento de imagem, protocolos de comunicação e Injetores de Aplicativos (AppInj). Em nossos experimentos precisamos fazer uso deste último, pois tem por função distribuir as tarefas a serem executados na região do GPPC. O funcionamento do AppInj é apresentado na Figura 19. Inicialmente a aplicação precisa ser decomposta em tarefas de forma a possibilitar a distribuição dos diversos blocos aos PEs que fazem parte da SoC.

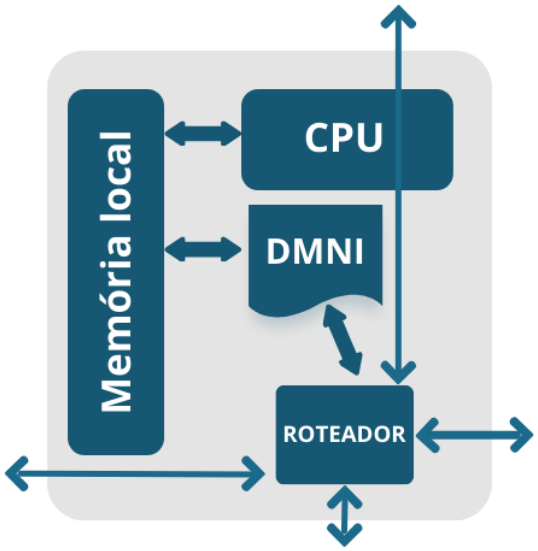


Figura 18: Elemento Processador (PE)

Para apoiar a injeção ou distribuição dinâmica das tarefas, é utilizado um protocolo que segue etapas definidas para determinar como será a distribuição das tarefas. Inicialmente, o processo começa no passo 1 com o Applnj solicitando a requisição de uma nova aplicação, enviando uma mensagem (*"NEW APP REQUEST"*) para um elemento processador mestre denominado MpE, localizado no *"cluster 0"* com o número de tarefas da aplicação. Este MpE faz a escolha de um *"cluster"* para executar a aplicação recebida de acordo com critério de recursos disponíveis, enviando uma mensagem *"APP ACK"* para o Applnj no passo 2, com o endereço do MpE localizado no ‘‘cluster"selecionado. No passo 3, o Applnj envia uma mensagem *"APP DESCRIPTOR"*, com as tarefas da aplicação que então são mapeadas. No passo 4 após o mapeamento de tarefas, o MpE envia uma mensagem *"APP ALLOCATION REQUEST"* para o Applnj, com os endereços dos PEs que receberão os códigos das tarefas. O Applnj então envia uma mensagem *"TASK ALLOCATION"* juntamente com o código referente a tarefa resgatado da memória via DMNI.

4.3.3 Diretivas de troca de mensagem

A troca de mensagens entre os PEs se dá com a utilização das diretivas de chamada de sistema denominadas de *SEND* e *RECEIVE*. Esta troca ocorre com a utilização de uma de mensagem denominada *Message* que possui uma variável *length* definindo o tamanho da mensagem, e um *array* de dados, chamado msg para os dados da mensagem. O processo tem suas etapas descritas na Figura 21 quando a diretiva *SEND* é acionada antes da diretiva *RECEIVE*. Inicialmente as tarefas A e B estão rodando respectivamente nos PEs

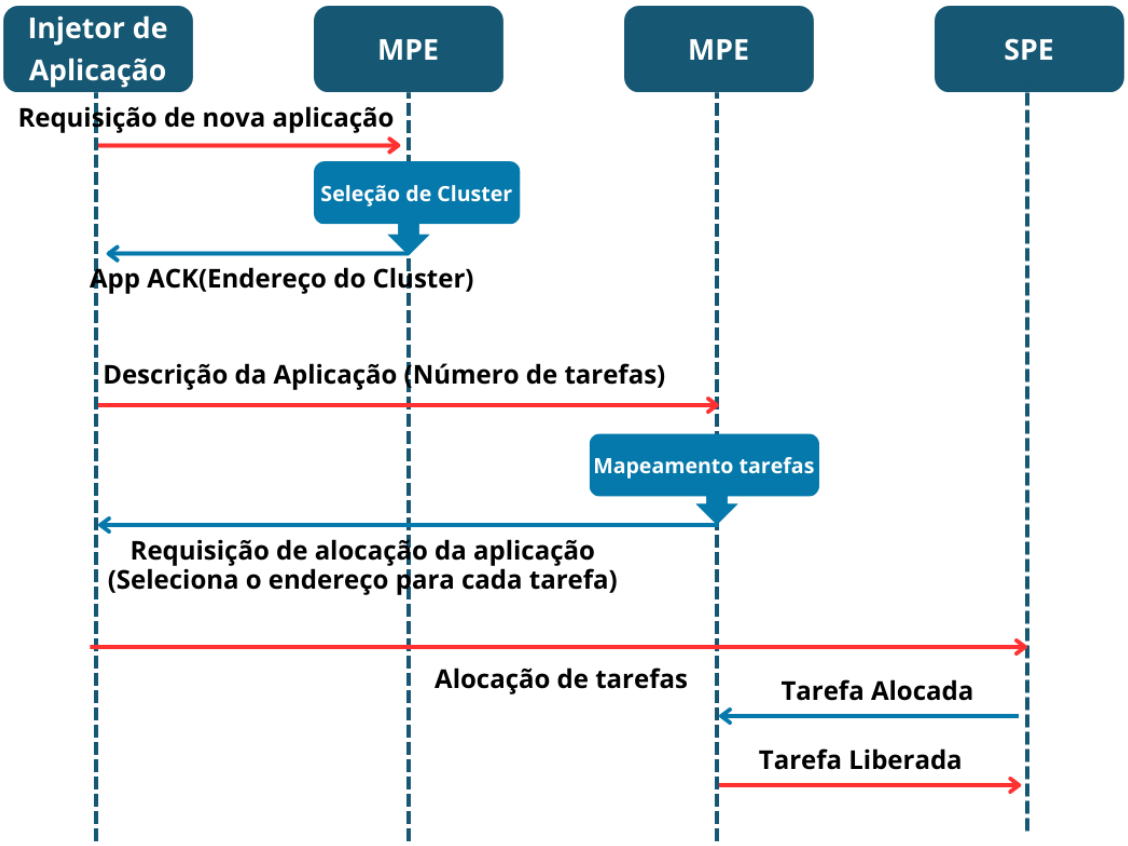


Figura 19: Distribuição de tarefas

1 e 2. Assim que ocorre um *SEND* a partir da tarefa B no PE2 tendo como destino a tarefa A no PE1 a diretiva *RECEIVE* ainda não foi gerada pela tarefa A. Enquanto a tarefa A não produzir a primitiva *RECEIVE*, a mensagem ficará armazenada no *buffer* do PE 2. Assim que a tarefa A executar o *RECEIVE* a mensagem que está armazenada no *buffer* será envia para o PE1 para que possa ser utilizado pela tarefa A.

Outra possibilidade é quando o PE1 que está com a tarefa A rodando, gera o *RECEIVE* antes da ocorrência do *SEND* conforme ilustra a Figura 21. Neste caso a tarefa A gera uma requisição chamada de *MESSAGE REQUEST* e não dará continuidade na execução até que seja gerado o *SEND* pela tarefa B. Assim que a tarefa B gera a primitiva *SEND* a mensagem produzida pela tarefa B é encaminhada diretamente para tarefa A.

4.3.4 Ferramenta de Debugger

A disponibilidade de ferramentas que viabilizam a checagem de desempenho em configurações específicas, ajudam na conclusão de quais experimentos apresentaram melhor resultado. A Figura 22, mostra a ferramenta *debugger* disponibilizada pela MEMPHIS

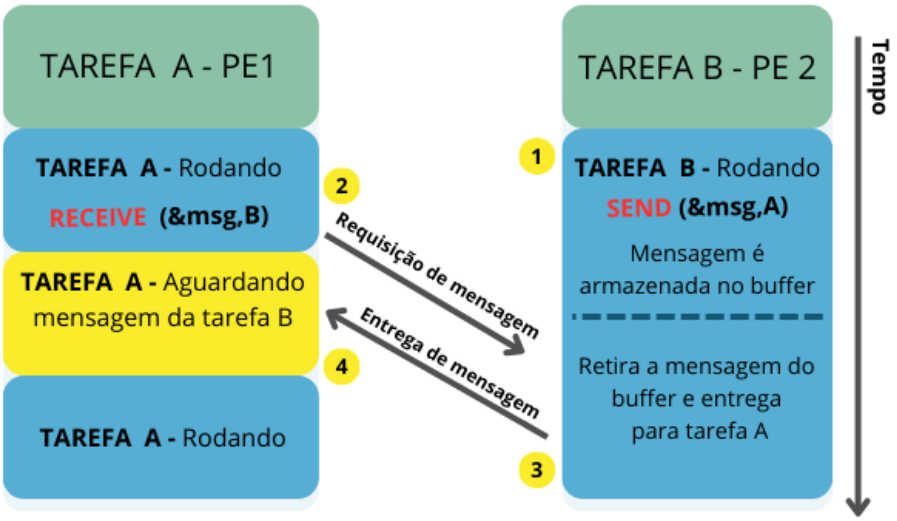


Figura 20: Troca de mensagem iniciando com SEND

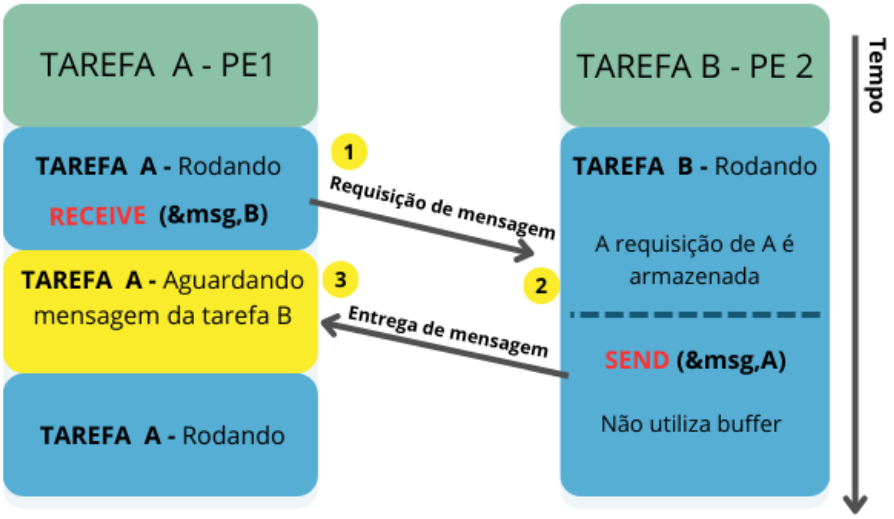


Figura 21: Troca de mensagem iniciando com RECEIVE

com este propósito. O Debugger disponibiliza outros recursos importantes na apuração referente ao andamento das tarefas que estão sendo executadas pelos PEs. A ferramenta Delorean, conforme mostra a Figura 23, foi disponibilizada com este propósito e apresenta um arquivo formato texto com informações pertinentes ao andamento das tarefas em execução.

4.4 Considerações Finais

Neste capítulo apresentamos o ambiente de experimentação, bem como conceitos importantes para o completo entendimento desta arquitetura multiprocessada. O que a

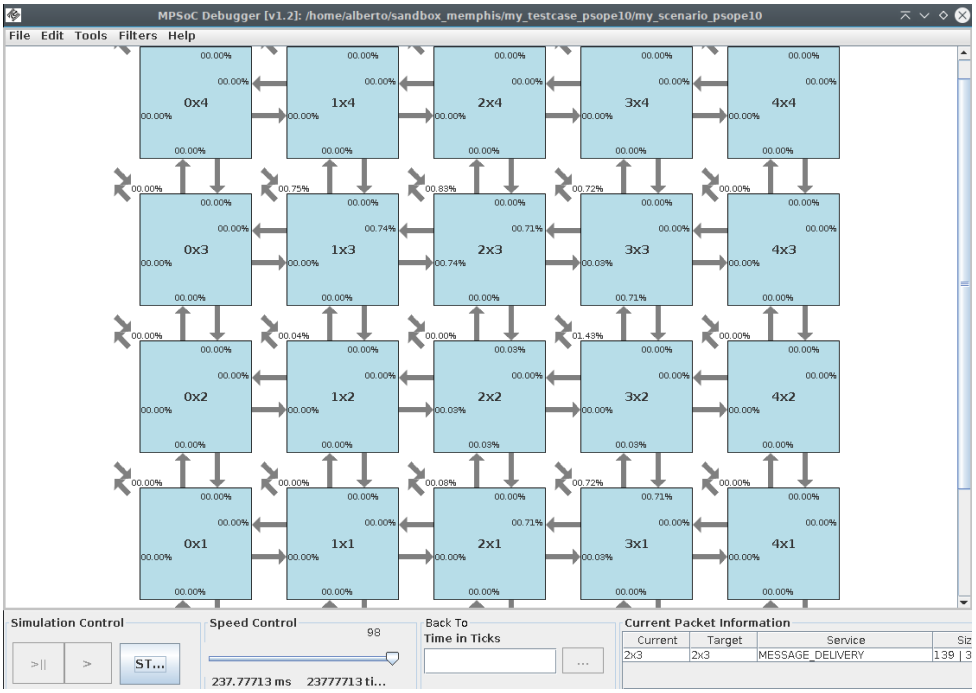


Figura 22: Ferramenta Debugger

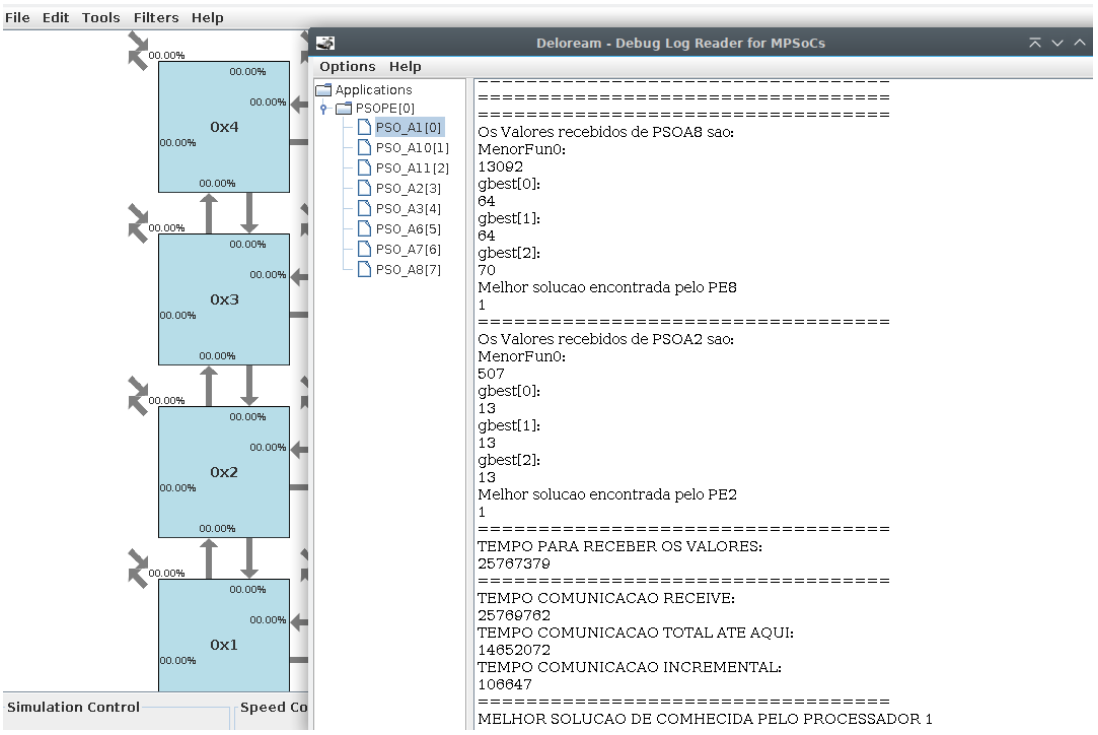


Figura 23: Ferramenta Delorean

plataforma oferece de melhor é a possibilidade de escalabilidade, permitindo assim a utilização de várias configurações de *hardware*, onde é possível testar diversas possibilidades de arranjos, topologias e número de processadores. Também de grande importância, é a disponibilidade da ferramenta Delorean que permite verificar o andamento das tarefas que

estão sendo executadas pelos PEs. É possível, por exemplo, incluir nos códigos diretivas, para que sejam exibidos o tempo gasto com as trocas de mensagens efetuada entre os PEs.

Capítulo 5

RESULTADOS EXPERIMENTAIS

NESTE capítulo, apresentamos os resultados experimentais obtidos por simulação, utilizando a plataforma MEMPHIS. Para efeito de comparação empregamos as mesmas configurações ao utilizarmos o PSO em sua forma serial e ao utilizarmos as estratégias paralelas. Para efeito de análise do ganho em desempenho, expresso em termos de *speedup*, fizemos a comparação entre os tempos de execução no modelo serial e no modelo paralelo. Além do tempo de execução, analisamos, também, o tempo de comunicação entre os processadores, no modelo paralelo. Isto permite identificar o impacto que a estratégia de paralelização empregada e o mapeamento adotado causam no *speedup* obtido. A Seção 5.1 apresenta as funções de teste utilizadas para implementação. A Seção 5.2 apresenta a divisão do espaço de busca. A Seção 5.3 apresenta aspectos da implementação. A Seção 5.4 apresenta os resultados utilizando a versão serial do PSO, a Seção 5.5 os resultados obtidos quando utilizamos as estratégias paralelas. A Seção 5.6 apresenta as considerações finais.

5.1 Funções de teste utilizadas

Para os testes comparativos referente à diferença de desempenho entre o modelo serial do PSO e as estratégias paralelas, uma das funções é a Rosenbrock (ROSENBROCK, 1960). Esta função é frequentemente usada para avaliar o desempenho de algoritmos de otimização, pois apresenta desafios, como uma superfície de busca com muitos mínimos locais. A função é definida conforme a Equação (5):

$$f(x) = \sum_{i=1}^{d} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]. \quad (5)$$

Em nossos experimentos estamos utilizando a função Rosenbrock com 3 variáveis, ou seja, d=3. Desta forma, substituindo o valor d na Equação (5) ficamos conforme a Equação (6)

$$f(x) = [100(x_2 - x_1^2)^2 + (x_1 - 1)^2] + [100(x_3 - x_2^2)^2 + (x_2 - 1)^2]. \quad (6)$$

A curva para esta função está ilustrado na Figura 24, sendo seu valor de mínimo igual a 0, na coordenada (1,1,1), para 3 dimensões, $f(x_1, x_2, x_3) = 0$.

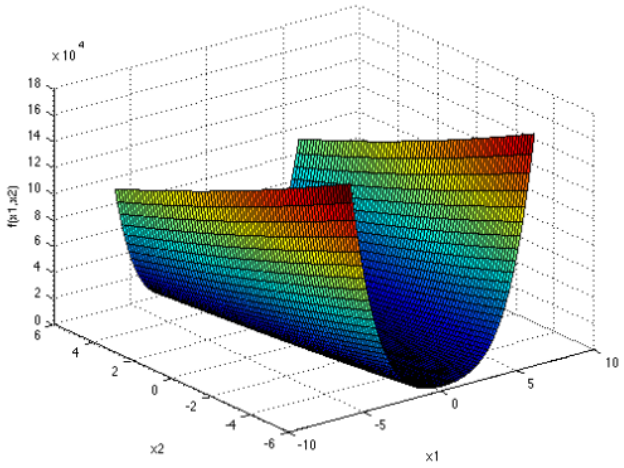


Figura 24: Função Rosenbrock. Fonte (SURVANOVICK, 2013)

.

Outra função que utilizamos é a Esfera (MOLGA; SMUTNICKI, 2005). É comumente utilizada em problemas de otimização, apesar de ser mais simples que a função Rosenbrock, sendo útil para avaliar o desempenho do algoritmos de otimização. Sua função é definida conforme a Equação (7):

$$f(x) = \sum_{i=1}^{d} x_i^2. \quad (7)$$

Para esta função também estamos utilizando em nossos experimento 3 variáveis, ou seja, d=3. Desta forma, substituindo o valor d na Equação (7) ficamos conforme a Equação (8)

$$f(x) = x_1^2 + x_2^2 + x_3^2. \quad (8)$$

A curva para a função Esfera está representada no gráfico da Figura 25. É uma função do tipo quadrática que apresenta o seu valor de mínimo igual a 0, na coordenada (0,0,0), com 3 dimensões, ou seja, $f(x_1, x_2, x_3) = 0$

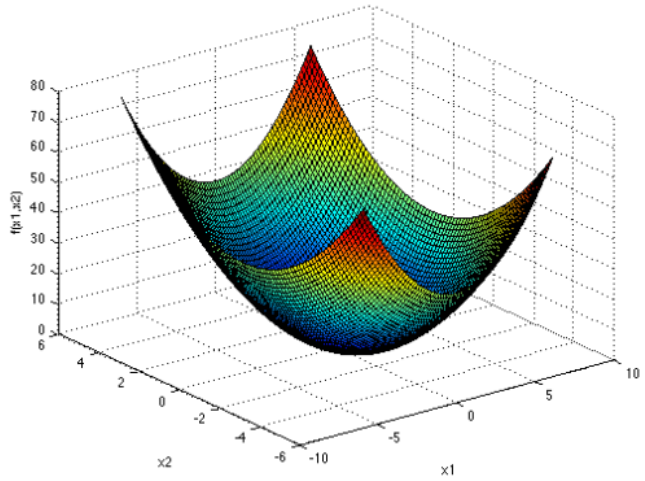


Figura 25: Função Esfera. Fonte (SURVANOVICK, 2013)

.

5.2 Definição do espaço de busca

Nos experimentos, utilizamos dois espaços de busca definidos respectivamente nos intervalos [0,50] e [0,100]. A estratégia escolhida quando utilizamos a divisão do espaço de busca foi manter cada processador PE dedicado apenas ao seu espaço. Cada um desses PEs terão informações a respeito dos melhores locais encontrados pelos seus vizinhos conforme a topologia utilizada e decidirão se devem ou não continuar o processo de otimização. Esta divisão no espaço de busca é realizada durante a inicialização do algoritmo executado em cada processador. O balanceamento de carga inicial, ou seja, a divisão do espaço de busca total em subespaços para cada processador será feito de forma proporcional. Considerando, por exemplo, o espaço [0,100] e 10 processadores teremos que cada subespaço por processador será dividido em porções de 10 partes iniciando com [0,9] para o processador 1 e finalizando com [90,99] para o processador 10. Para melhor ilustrar apresentamos as tabelas de particionamento do espaço de busca conforme número de processadores utilizados, que variam de 2 até 12.

A coluna "Divisão do Espaço de busca", apresenta os espaços destinados a cada processador durante o processo de otimização. O intervalo varia de Espaço 1 (*Esp1*) até Espaço 2 (*Esp2*) conforme o número de processadors (PEs).

Tabela 1: Divisão do espaço de busca para o intervalo [0,50]

Divisão	Número PEs				
Espaço de busca	2	4	8	10	12
$Esp1$	$[0, 24]$	$[0, 11]$	$[0, 6]$	$[0, 4]$	$[0, 3]$
$Esp2$	$[25, 50]$	$[12, 22]$	$[7, 12]$	$[5, 9]$	$[4, 7]$
$Esp3$		$[23, 35]$	$[13, 18]$	$[10, 14]$	$[8, 11]$
$Esp4$		$[36, 50]$	$[19, 24]$	$[15, 19]$	$[12, 15]$
$Esp5$			$[25, 30]$	$[20, 24]$	$[16, 19]$
$Esp6$			$[31, 36]$	$[25, 29]$	$[20, 23]$
$Esp7$			$[37, 43]$	$[30, 34]$	$[24, 27]$
$Esp8$			$[44, 50]$	$[35, 39]$	$[28, 32]$
$Esp9$				$[40, 44]$	$[33, 35]$
$Esp10$				$[45, 50]$	$[36, 39]$
$Esp11$					$[40, 44]$
$Esp12$					$[45, 50]$

Tabela 2: Divisão espaço de busca para o espaço [0,100]

Divisão	Número PEs				
Espaço de busca	2	4	8	10	12
$Esp1$	$[0, 49]$	$[0, 24]$	$[0, 12]$	$[0, 09]$	$[0, 7]$
$Esp2$	$[50, 100]$	$[25, 49]$	$[13, 25]$	$[10, 19]$	$[8, 15]$
$Esp3$		$[50, 74]$	$[26, 38]$	$[20, 29]$	$[16, 23]$
$Esp4$		$[75, 100]$	$[39, 51]$	$[30, 39]$	$[24, 31]$
$Esp5$			$[52, 63]$	$[40, 49]$	$[32, 39]$
$Esp6$			$[64, 75]$	$[50, 59]$	$[40, 47]$
$Esp7$			$[76, 87]$	$[60, 69]$	$[48, 55]$
$Esp8$			$[88, 100]$	$[70, 79]$	$[56, 63]$
$Esp9$				$[80, 89]$	$[64, 72]$
$Esp10$				$[90, 100]$	$[73, 81]$
$Esp11$					$[82, 90]$
$Esp12$					$[91, 100]$

5.3 Aspectos de Implementação

Para os experimentos foram utilizadas 200 partículas e um espaço de busca tridimensional, definido pelas variáveis $x1$, $x2$ e $x3$. Para a função Rosenbrock definimos o espaço de busca no intervalo [0, 50] e para a função esfera utilizamos 2 espaços de busca [0, 50] e [0, 100] respectivamente. Para que ocorra uma distribuição balanceada das partículas em seus respectivos espaços de busca, o enxame inicial será dividido em subenxames

proporcionais ao número de processadores, ou seja, considerando n a quantidade de partículas do enxame e $nproc$ o número de processadores, teremos em cada subespaço de busca aproximadamente $n/nproc$ partículas. Estas partículas serão geradas sempre de forma aleatória a cada experimento. Conforme (WAINTRAUB; SCHIRRU; PEREIRA, 2009), durante a execução do PSO com a continuidade no deslocamento das partículas, estas tendem a ultrapassar o espaço de busca delimitado. Para contornar esta característica, foi implementado uma função, que reposiciona de forma randômica a partícula em seu espaço de busca.

As variáveis r_1 e r_2 produzem valores aleatórios entre 0 ou 1 conforme abordamos no Capítulo 1. Este fator é multiplicado aos termos relacionados a melhor posição local e global das partículas e permite acrescentar um grau de aleatoriedade no peso destes fatores, o que pode ajudar na exploração do espaço de busca.

Conforme visto no Capítulo 1, os melhores valores para fator de inércia w e para os coeficientes de aprendizagem individual c_1 e global c_2, dependem do problema específico que está sendo otimizado. Estes parâmetros foram ajustados empiricamente durante a fase de experimentação e foram definidos da seguinte forma: para o fator de inércia w utilizamos o valor 1 e os coeficientes de aprendizagem individual e global definidos com o valor 2, ou seja, $c_1 = c_1 = 2$.

Os dados obtidos foram consolidados em tabelas que apresentam o tempo de execução T_S em milissegundos (ms), necessário para que a primeira partícula do enxame alcançar o valor de mínimo nas funções Rosenbrock e Esfera, bem como o número de iterações necessárias e o tempo de comunicação entre os processadores.

5.4 Resultados da versão serial do PSO

Os dados referentes ao experimento do PSO utilizando a versão serial inerente ao algoritmo, estão apresentados na Tabela 3. As funções de teste utilizadas foram a Rosenbrock e Esfera conforme citado na seção 5.1 sendo considerado o espaço de busca total definidos nos intervalos [0,50] para a função Rosenbrock e [0,50] e [0,100] para a função Esfera. A condição de parada estipulada será quando a primeira partícula do enxame identificar o valor de mínimo global. A Tabela 3 apresenta o resultado de 10 simulações feitas com o objetivo de reduzir a diferença entre os valores de durações obtidos em cada uma, em função do efeito estocástico do cálculo. Ao final da tabela apresentamos o valor do tempo

médio T_S para cada uma das funções. Obtivemos como média T_S para este experimento o tempo de execução de 195,88 ms e 628 iterações quando utilizamos a função Rosenbrock no espaço de busca [0,50]. Para a função Esfera obtivemos tempo de execução de 87,39 ms e 292 iterações para o espaço de busca [0,50] e 189,91 ms e 667 iterações para o espaço de busca [0,100].

Tabela 3: Número de iterações e tempo de execução da versão serial do PSO

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	1230	364,44	163	49,72	214	60,56
Simulação 2	336	101,11	105	32,20	882	243,95
Simulação 3	548	164,80	164	50,71	1213	336,92
Simulação 4	2177	634,85	413	123,49	643	179,47
Simulação 5	496	148,03	777	228,23	1037	288,42
Simulação 6	601	261,72	576	170,83	139	42,07
Simulação 7	100	31,61	233	71,19	779	228,66
Simulação 8	301	90,63	45	14,99	645	189,73
Simulação 9	263	79,42	352	104,66	269	80,40
Simulação 10	270	82,17	84	26,94	847	248,89
Média	628	195,88	292	87,39	667	189,91

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 4, sendo de 173,84 ms, 65,79 ms e 94,81 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 4: Valores de duração média e desvio padrão para versão serial do PSO

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	195,88	87,39	189,91
Desvio padrão (ms)	173,84	65,79	94,81

5.5 Resultados da versão paralela do PSO

Apresentaremos nesta seção os resultados obtidos ao utilizarmos a estratégia paralela variando as topologias de comunicação e a quantidade de processadores. Um dos princi-

pais indicadores de desempenho que utilizaremos será o valor de *speedup* que será obtido conforme a Equação (9):

$$speedup = T_S/T_P, \tag{9}$$

onde T_S corresponderá ao valor médio referente ao tempo de otimização necessário para execução do PSO na forma serial e o valor T_p ao valor médio referente ao tempo de otimização necessário para execução do PSO na forma paralela. Tanto para a versão serial quanto para a versão paralela as médias serão obtidas a partir da execução de 10 simulações.

Na Seção 5.5.1 iniciaremos os experimentos fazendo uso do paralelismo utilizando a topologia Mestre-trabalhador. Na Seção 5.5.2 a topologia em Anel e finalmente na Seção 5.5.3 a topologia em Malha 2D.

5.5.1 Topologia Mestre-trabalhador

Iniciamos as experimentações em paralelo, utilizando a estratégia mestre-trabalhador e 2 processadores. Na sequência, aumentamos este número para 4,8,10 e 12 processadores. Conforme fizemos nos experimentos para a versão serial também faremos 10 simulações para minimizar o efeito estocástico do cálculo nas versões paralelas. Em todos os experimentos as partículas foram geradas aleatoriamente dentro do espaço de busca definido para cada processador, conforme descrito na Seção 5.2.

5.5.1.1 Topologia Mestre-trabalhador utilizando 2 processadores

Podemos verificar os resultados experimentais obtidos quando utilizamos 2 processadores na Tabela 5. Obtivemos como valor médio o tempo de execução de 108,33 ms, 640 iterações e *speedup* de 1,80 quando utilizamos a função Rosenbrock. Para a função Esfera obtivemos o tempo médio de execução de 76,53 ms, 444 iterações para o espaço de busca [0,50] e *speedup* de 1,14. E para a função Esfera no espaço de busca [0,100] o tempo médio de 76,47 ms, 460 iterações e um *speedup* de 2,48. Podemos verificar a partir desta tabela a obtenção de valores superlineares de *speedups*, que ocorre quando o desempenho do sistema aumenta em uma taxa maior do que a esperada com base no número de processadores. Esta resultado pode ser explicado pela diminuição de partículas que ocorre juntamente

com o aumento de processadores, ou seja, cada novo processador inserido no experimento fica responsável por um número cada vez menor de partículas.

Tabela 5: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 2 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	417	70,10	1364	231,46	196	33,09
Simulação 2	117	2034	84	84	943	155,99
Simulação 3	205	36,52	67	14,04	1701	280,94
Simulação 4	545	94,35	573	98,90	282	47,32
Simulação 5	1912	329,65	354	61,65	345	57,70
Simulação 6	289	50,51	215	37,34	104	18,17
Simulação 7	383	66,74	277	47,89	175	29,76
Simulação 8	482	83,24	778	133,01	429	71,21
Simulação 9	437	75,79	420	72,27	273	45,74
Simulação 10	1491	256,02	302	52,29	144	24,74
Média	640	108,33	444	76,53	460	76,47
speedup	1,80		1,14		2,48	

O mapeamento das tarefas foi realizado de forma estática na ferramenta de experimentação e ficou conforme a Figura 26, que também ilustra como está ocorrendo a troca de mensagens, utilizando as diretivas SEND e RECEIVE apresentadas no Capítulo 4.

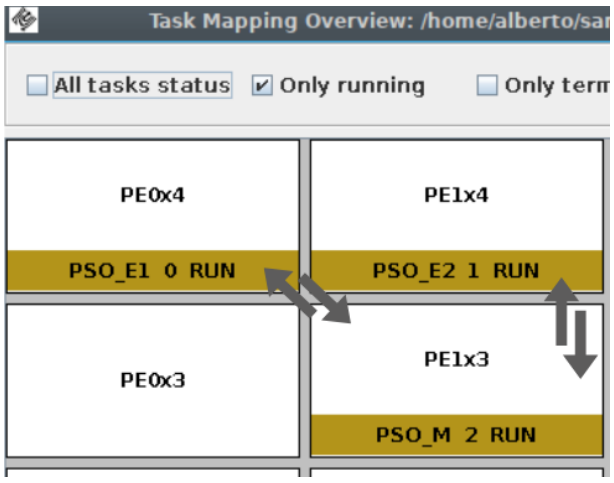


Figura 26: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Mestre-trabalhador utilizando 2PEs

A figura indica através de setas como ocorre o fluxo de troca de mensagens. Como neste caso estamos utilizando 2 PEs, também dispomos de 2 tarefas que estão identificadas com os nomes PSO A1 e PSO A2, enquanto uma tarefa PSO M é executada pelo PE mestre, os PEs trocam mensagens apenas com o PE Mestre. As tarefas PSO A1 e PSO A2, executam seus códigos em linguagem C para o PSO, explorando cada um seus respectivos espaços de busca durante a otimização.

Para o cálculo do desvio padrão das durações referentes as 10 simulações, obtivemos os resultados conforme a Tabela 6, sendo de 95,97 ms, 61,95 ms e 77,74 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 6: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando 2PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	108,33	76,53	76,47
Desvio padrão (ms)	95,97	61,95	77,74

Os tempos referente à troca de mensagem entre os dois PEs e o PE Mestre, estão listados na Tabela 7. Para o levantamento utilizamos os valores médios obtidos a partir da Tabela 5.

Tabela 7: Tempo médio, das 10 simulações, gasto com troca de mensagem na versão paralela do PSO com topologia Mestre-trabalhador utilizando 2 PEs

	Tempo médio (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Envio parâmetros para PEs	0,0151	0,0160	0,0150
PE Mestre escolha melhor solução	0,0729	0,0376	0,0170
Atualização PEs melhor resultado	13,1524	13,0803	9,6202
Tempo total troca de mensagem	13,2404	13,3039	9,6522

A primeira linha mostra o tempo médio gasto pelo processador mestre para enviar os parâmetros iniciais do PSO para os PEs trabalhadores. A segunda linha indica o tempo gasto pelo PE Mestre na escolha da melhor solução que será o ótimo global. Este valor corresponde ao tempo gasto ao final do processo para que o PE Mestre selecione o melhor valor de ótimo local entre os enviados pelos PEs trabalhadores. O tempo referente à troca

de mensagem que ocorre entre o PE Mestre e os PEs trabalhadores, com a atualização do melhor resultado a cada iteração, consta na terceira linha, sendo de 0,020925 ms a cada iteração quando utilizamos 2 PEs. Conforme o aumento do número de PEs teremos também incremento deste valor. O total de tempo gasto com a atualização dos PEs trabalhadores com o melhor resultado de ótimo local será obtido multiplicando o valor 0,020925 ms pelo número de iterações necessárias em cada experimento. A última linha nesta tabela é o tempo total gasto em troca de mensagem obtido a partir do somatório dos dados das 3 primeiras linhas.

Faremos agora uma comparação dos tempos médios obtidos com troca de mensagem com o tempo total do experimento. Quando utilizamos a função Rosenbrock o tempo de mensagens representa 12,24% do tempo total do experimento, para a função esfera no espaço de busca [0,50] este valor ficou em 16,73% e finalmente para a função esfera no espaço de busca [0,100] este valor ficou em 12,51%. Pode-se concluir a partir destes resultados que o tempo de mensagem gasto a cada iteração com a atualização dos PEs trabalhadores é o que impacta mais neste tipo de topologia.

5.5.1.2 Topologia Mestre-trabalhador utilizando 4 processadores

Nesta etapa fizemos um incremento de 2 PEs totalizando 4 PEs para experimentação. Conforme aumentamos o número de PEs ocorre a diminuição no espaço de busca definido para cada processador conforme apresentado na Seção5.2. Os dados obtidos estão na Tabela 8 e as médias na penúltima linha. Podemos observar novamente valores de *speedups* superlineares.

Para a função Rosenbrock encontramos o Tempo de execução de 5,58 ms e 51 iterações. Para a função Esfera obtivemos o tempo de execução de 2,41 ms e 16 iterações para o espaço de busca [0,50] 52,48 ms e 559 iterações para o espaço de busca [0,100]. Os valores de *speedup* constam na última linha desta tabela.

O mapeamento das tarefas, na ferramenta de experimentação de simulação, ficou conforme a Figura 27. Neste experimento estamos utilizando 4 PEs com suas respectivas 4 tarefas identificadas com os nomes PSO E1, PSO E2, PSO E3 e PSO E4 e o processador mestre PSO M.

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a a Tabela 9, sendo de 5,26 ms, 1,10 ms e 45,15 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 8: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 4 PEs.

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	1	0,52	47	5,18	11	1,60
Simulação 2	12	1,67	11	1,93	471	43,80
Simulação 3	56	5,85	23	3,14	507	47,15
Simulação 4	70	7,33	5	1,49	1605	148,05
Simulação 5	197	19,46	6	1,67	45	4,76
Simulação 6	63	6,99	15	2,45	430	41,02
Simulação 7	26	3,52	5	1,44	1109	104,65
Simulação 8	8	1,85	15	2,48	877	82,90
Simulação 9	65	7,27	4	1,37	516	48,37
Simulação 10	3	1,39	21	2,92	19	2,54
Média	51	5,58	16	2,41	559	52,48
speedup	35,04		36,23		3,61	

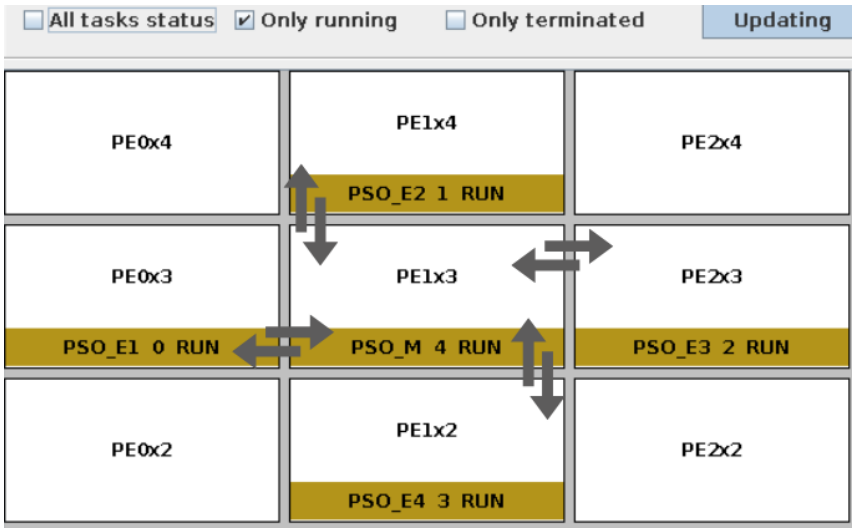


Figura 27: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Mestre-trabalhador utilizando 4PEs

Os dados referentes a troca de informação entre os PEs foram levantados com a mesma metodologia utilizada para 2PEs. Os resultados estão na Tabela 10. O tempo referente a troca de mensagem que ocorre a cada iteração entre o PE Mestre e os PEs trabalhadores, ficou em 0,05219 ms, sendo esta duração multiplicada pelo número de iterações na experimentação. Podemos observar que os resultados obtidos de tempo total de troca de mensagens foram menores quando comparados com a versão utilizando 2

Tabela 9: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando 4PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	5,58	2,41	52,48
Desvio padrão (ms)	5,26	1,10	45,15

PEs. Este fato ocorre devido esse tempo total depender do número de iterações. Como para 4 PEs chegamos ao término do experimento com um número menor de iterações consequentemente o tempo de total de troca de mensagens foi menor.

Agora faremos a análise dos tempos percentuais médio de troca de mensagem quando comparamos com a duração total do experimento. Para a função Rosenbrock o tempo de troca de mensagem representa 40,32% do tempo total do experimento, para a função esfera no espaço de busca [0,50] este valor ficou em 36,20% e em 51,86% para a função esfera no espaço de busca [0,100]. Pode-se observar o aumento médio do tempo de troca de mensagem conforme o aumento do múmero de processadores e iterações.

Tabela 10: Tempo médio, das 10 simulações, gasto com troca de mensagem na versão paralela do PSO com topologia Mestre-trabalhador utilizando 4 PEs.

	Tempo médio (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Envio parâmetros para PEs	0,019	0,019	0,019
PE Mestre escolha melhor solução	0,0602	0,1332	0,0174
Atualização PEs melhor resultado	2,6147	0,7932	29,1741
Tempo total troca de mensagem	2,6939	0,9499	29,9106

5.5.1.3 Topologia Mestre-trabalhador utilizando 8 processadores

Utilizando 8 processadores os dados obtidos estão apresentados na Tabela 11. As médias para o tempo de execução utilizando a função Rosenbrock foi de 3,31 *ms* e 31 iterações. Para a função Esfera obtivemos o tempo de execução de 1,17 *ms* e 11 iterações para o espaço de busca [0,50] e 22,48 *ms* e 271 iterações para o espaço de busca [0,100]. Os valores de *speedup* estão na última linha da tabela.

Tabela 11: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 8 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	4	0,67	25	2,68	19	1,35
Simulação 2	81	7,88	7	0,99	324	19,07
Simulação 3	37	3,91	36	3,54	35	2,26
Simulação 4	53	5,31	1	0,62	416	23,05
Simulação 5	48	4,84	2	0,55	438	24,04
Simulação 6	25	3,17	1	0,40	441	45,98
Simulação 7	1	0,44	8	1,35	298	31,25
Simulação 8	8	1,394	6	1,12	266	27,93
Simulação 9	43	5,05	12	1,75	181	19,19
Simulação 10	1	0,41	9	1,45	292	30,63
Média	31	3,31	11	1,17	271	22,48
speedup	59,15		74,06		8,44	

O mapeamento das tarefas, na ferramenta de experimentação de simulação, ficou conforme a Figura 28. As 8 tarefas foram identificadas com os nomes PSOE1 até PSOE8 além do PE mestre.

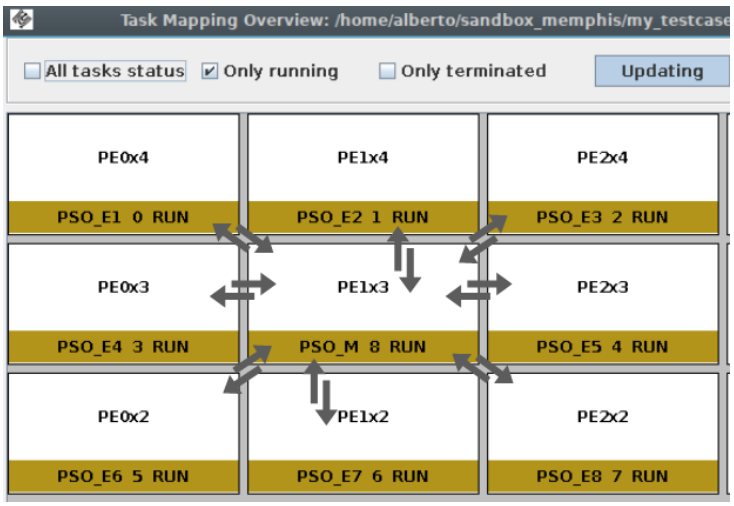


Figura 28: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Mestre-trabalhador utilizando 8PEs

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a a Tabela 12, sendo de 2,40 ms, 0,94 ms e 12,50 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 12: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando 8 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	3,31	1,17	22,48
Desvio padrão (ms)	2,40	0,94	12,50

A Tabela 13 traz os resultados para o tempo de troca de mensagem. Considerando os 8PEs, o tempo referente a troca de mensagem a cada iteração que ocorre entre o PE Mestre e os PEs trabalhadores, ficou em 0,6235 ms para as funções Rosenbrok e Esfera [0,50] e 0,0554 ms para a função Esfera [0,100] . Para a função Rosenbrock o tempo de troca de mensagem representa 49,54% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 68,82% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 72,91%. Como o tempo médio para otimizar as funções diminui significativamente, o tempo médio de troca de mensagem passa a representar um valor maior no tempo total.

Tabela 13: Tempo médio, das 10 simulações, gasto com troca de mensagem na versão paralela do PSO com topologia Mestre-trabalhador utilizando 8 PEs

	Tempo médio (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Envio parâmetros para PEs	0,03878	0,03888	0,03390
PE Mestre escolha melhor solução	0,016	0,2704	0,016
Atualização PEs melhor resultado	1,8767	0,6671	15,0134
Tempo total troca de mensagem	1,9315	0,9763	15,0633

5.5.1.4 Topologia Mestre-trabalhador utilizando 10 processadores

Os dados obtidos estão apresentados na Tabela 14. Os resultados das médias para o tempo de execução utilizando a função Rosenbrock foi de 0,61 ms e 2 iterações. Para a função Esfera obtivemos o Tempo de execução de 0,56 ms e 3 iterações para o espaço de busca [0,50] e 9,65 ms e 82 iterações para o espaço de busca [0,100]. Os valores de *speedup* estão na última linha.

Tabela 14: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 10 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	1	0,54	1	0,46	68	7,14
Simulação 2	2	0,64	9	1,30	150	15,07
Simulação 3	1	0,39	1	0,37	106	10,86
Simulação 4	1	0,40	1	0,28	26	2,91
Simulação 5	1	0,38	1	0,60	45	4,83
Simulação 6	4	1,11	2	0,53	68	9,11
Simulação 7	3	0,70	1	0,31	12	1,83
Simulação 8	1	0,34	2	0,53	3	0,68
Simulação 9	2	0,57	1	0,39	2	0,53
Simulação 10	6	1,09	4	0,78	332	43,53
Média	2	0,61	3	0,56	82	9,65
speedup	315,78		154,01		19,67	

O mapeamento das tarefas, na ferramenta de experimentação de simulação, ficou conforme a Figura 29. As 10 tarefas foram identificadas com os nomes PSOE1 até PSOE10 além do PE mestre.

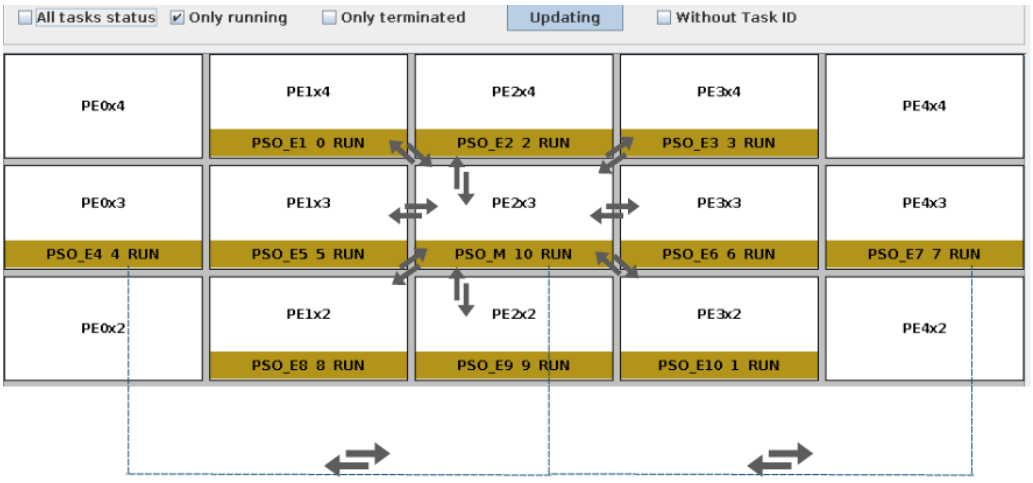


Figura 29: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Mestre-trabalhador utilizando 10PEs

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 15, sendo de 0,26 ms, 0,28 ms e 12,16 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 15: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando 10PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	0,61	0,56	9,65
Desvio padrão (ms)	0,26	0,28	12,16

A Tabela 16 traz os resultados para o tempo de troca de mensagem. Utilizando 10 PEs o tempo gasto por iteração para atualização dos PEs trabalhadores ficou em 0,1139 ms para as funções Rosenbrok e Esfera [0,50] e 0,0856 ms para a função Esfera [0,100]. Para a função Rosenbrock o tempo de troca de mensagem representa 54,16% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 77,50% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 85,32%.

Tabela 16: Tempo médio, das 10 simulações, gasto com troca de mensagem na versão paralela do PSO com topologia Mestre-trabalhador utilizando 10 PEs

	Tempo médio (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Envio parâmetros para PEs	0,04627	0,04591	0,04631
PE Mestre escolha melhor solução	0,04465	0,1267	0,3453
Atualização PEs melhor resultado	0,2505	0,2619	6,9531
Tempo total troca de mensagem	0,3415	0,4345	7,3447

5.5.1.5 Topologia Mestre-trabalhador utilizando 12 processadores

Finalmente no último experimento utilizando paralelismo na topologia Mestre-trabalhador, passamos a utilizar 12 PEs. Os dados obtidos estão apresentados na Tabela 17.

Os resultados médios para o tempo de execução utilizando a função Rosenbrock foi de 8,01 ms e 54 iterações. Para a função Esfera obtivemos o Tempo de execução de 9,56 ms e 69 iterações para o espaço de busca [0,50] e de 38,37 ms e 251 iterações para o espaço de busca [0,100].

O mapeamento das tarefas, na ferramenta de experimentação de simulação, ficou conforme a Figura 30. As 12 tarefas foram identificadas com os nomes PSOE1 até PSOE12 além do PE mestre.

Tabela 17: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 12 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	79	12,12	95	14,02	388	51,74
Simulação 2	1	0,35	47	7,06	4	0,80
Simulação 3	167	25,18	1	0,34	271	39,87
Simulação 4	1	0,32	17	0,72	116	17,11
Simulação 5	69	10,42	218	32,06	260	38,20
Simulação 6	3	0,65	77	11,05	99	14,75
Simulação 7	5	0,87	126	15,77	652	101,49
Simulação 8	9	1,51	94	13,45	69	11,02
Simulação 9	147	21,39	5	0,85	246	38,44
Simulação 10	53	7,89	1	0,28	690	107,30
Média	54	8,01	69	9,56	251	38,37
speedup	24,45		9,13		4,94	

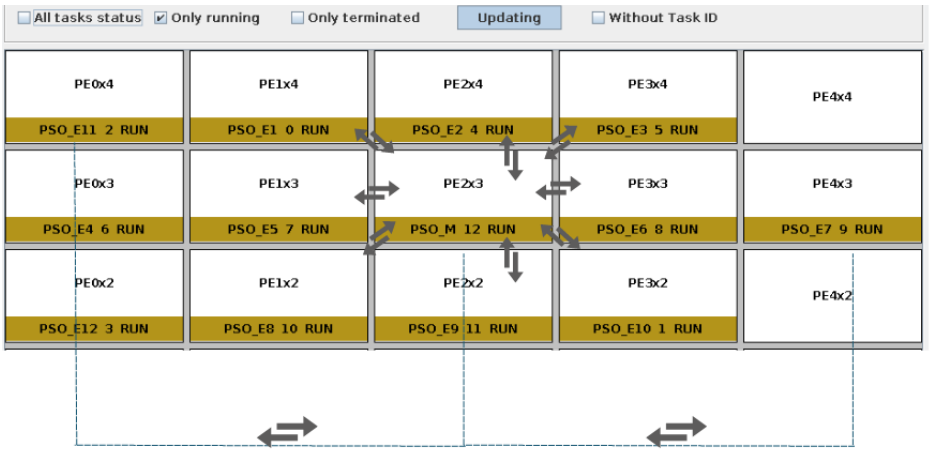


Figura 30: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Mestre-trabalhador utilizando 12PEs

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 18, sendo de 8,72 ms, 9,53 ms e 34,56 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

A Tabela 19 traz os resultados para o tempo gasto com troca de mensagem. O tempo médio gasto por iteração para que o PE Mestre atualize os PEs trabalhadores com o melhor valor ficou em 0,1254 ms. Para a função Rosenbrock o tempo de troca de mensagem representa 78,74% do tempo total do experimento, para a função Esfera no

Tabela 18: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando 12 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	8,01	9,56	38,37
Desvio padrão (ms)	8,72	9,53	34,56

espaço de busca [0,50] este valor ficou em 96,94% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 74,79%.

Tabela 19: Tempo médio, das 10 simulações, gasto com troca de mensagem na versão paralela do PSO com topologia Mestre-trabalhador utilizando 12 PEs

	Tempo médio (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Envio parâmetros para PEs	0,0552	0,0539	0,0571
PE Mestre escolha melhor solução	0,0391	0,2736	0,0246
Atualização PEs melhor resultado	6,6963	8,3987	35,0493
Tempo total troca de mensagem	6,7906	8,7262	35,131

Com a utilização de 12 processadores observamos que ocorreu grande perda de *speedup* quando comparamos com as versões com menos PEs. Tal fato ocorre pois apesar da redução do espaço de busca com o aumento do número de PEs ocorre proporcionalmente a redução do número de partículas , dificultando o processo de otimização. Em nossos experimentos encontramos este limite utilizando 10 PEs. Para validar esta conclusão, fizemos simulações com 400 partículas para a função Rosenbrock variando o múmero de processadores de 2 até 12. Os resultados estão na Tabela 20 para 2 , 4 e 8 processadores e a Tabela 21 para 10 e 12 processadores.

Observamos que utilizando 400 partículas e variando o número de PEs entre 4 e 12, os resultados de *speedup* apresentaram ganho significativos quando comparamos com os valores obtidos na versão utilizando 200 partículas. Somente quando utilizamos dois PEs é que o resultado foi inferior. Este fato deve-se ao esforço computacional necessário para executar o PSO com 400 partículas, não compensar quando utilizamos um número reduzido de PEs. Na Seção 5.5.1.6 apresentaremos um comparativo gráfico levando em-consideração a variação entre número de processadores e

Tabela 20: Número de iterações e tempo de execução com 400 partículas para a função Rosenbrock.

	Rosenbrock [0,50] 2 PEs		Rosenbrock [0,50] 4 PEs		Rosenbrock [0,50] 8 PEs	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	727	239,50	4	2,30	50	1,04
Simulação 2	874	286,847	11	3,47	1	0,75
Simulação 3	503	36,52	166,52	9	3	1,37
Simulação 4	95	33,23	11	3,72	71	7,96
Simulação 5	117	40,89	197	5,21	1	0,80
Simulação 6	268	88,26	7	2,87	2	1,30
Simulação 7	16	7,65	77	2,13	1	0,90
Simulação 8	57	20,77	1	1,50	1	0,77
Simulação 9	65	23,51	1	1,45	1	0,75
Simulação 10	379	125,92	15	4,28	1	0,80
Média	311	103,31	34	3,01	14	1,64
speedup	1,89		64,93		118,87	

Tabela 21: Número de iterações e tempo de execução da versão paralela do PSO com topologia Mestre-trabalhador utilizando 10 e 12 PEs com 400 partículas e função Rosenbrock.

	Rosenbrock [0,50] 10 PEs		Rosenbrock [0,50] 12 PEs	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	1	1,41	1	0,40
Simulação 2	5	1,09	5	1,11
Simulação 3	5	1,36	5	1,24
Simulação 4	1	1,01	1	0,76
Simulação 5	1	0,79	1	0,76
Simulação 6	1	0,59	53	8,14
Simulação 7	3	1,02	5	1,31
Simulação 8	3	1,02	31	5,00
Simulação 9	1	0,53	10	2,04
Simulação 10	1	0,53	4	1,15
Média	3	0,94	12	2,19
speedup	208,35		89,22	

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 22, sendo de 93,86 ms, 1,4 ms, 2,11 ms, 0,30 ms e 2,33 ms para a função Rosenbrock quando variamos o número de PES de dois até doze.

Tabela 22: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Mestre-trabalhador utilizando de 2 até 12 PEs e a função Rosenbrock

	Rosenbrock [0,50]				
Número de processadores	2	4	8	10	12
Média (ms)	103,31	3,01	1,64	0,94	2,19
Desvio padrão (ms)	93,86	1,4	2,11	0,30	2,33

5.5.1.6 Análise comparativa *speedup* estratégia Mestre-trabalhador

Esta seção traz um comparativo gráfico quando utilizamos paralelismo na topologia Mestre-trabalhador variando o número de processadores. Para a função Rosenbrock o gráfico está na Figura 31. Para a função Esfera no intervalo [0,50] e Esfera [0,100] os gráficos estão respectivamente nas Figuras 32 e 33.

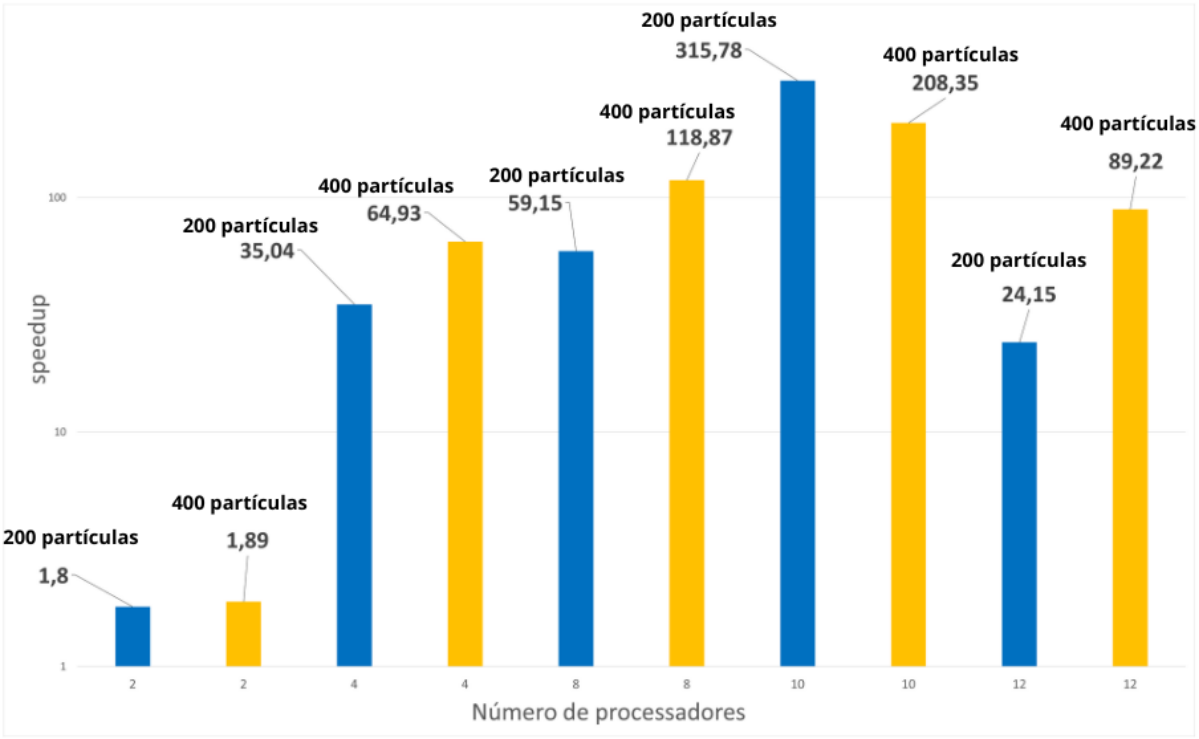


Figura 31: Speedup x número de processadores Função Rosenbrock - Topologia Mestre-trabalhador

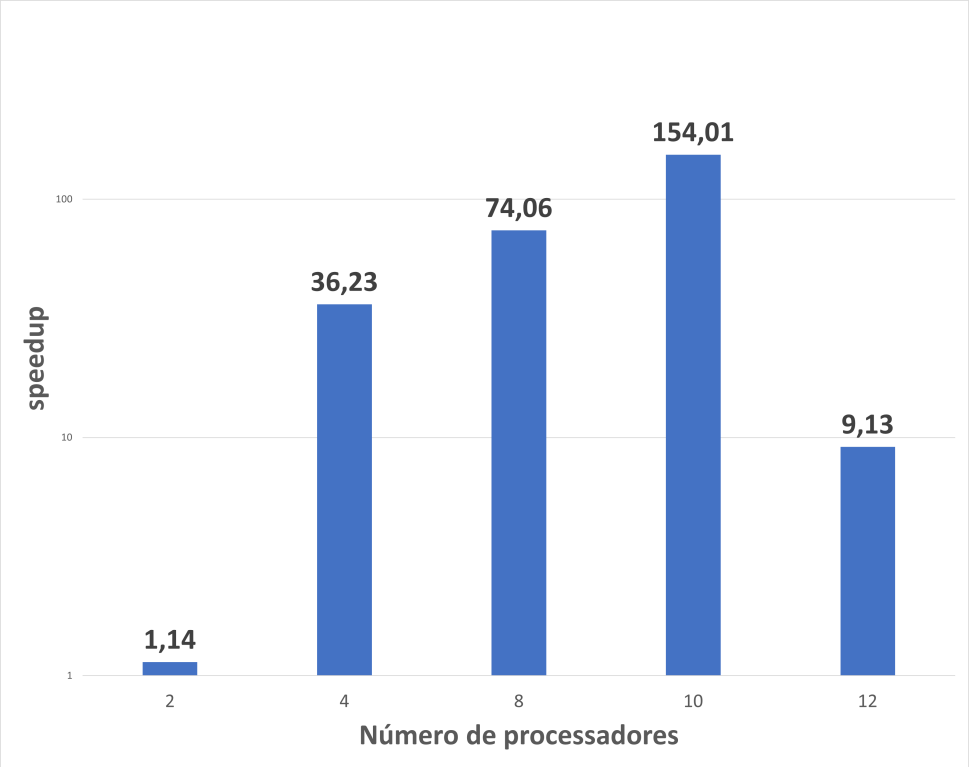


Figura 32: Speedup x número de processadores Função Esfera [050] - Topologia Mestre-trabalhador

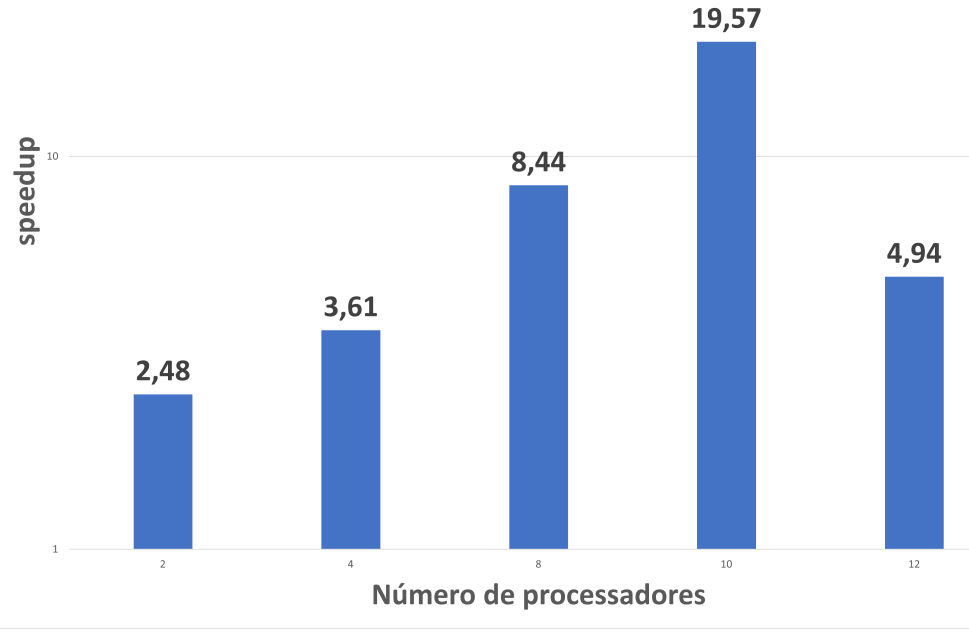


Figura 33: Speedup x número de processadores Função Esfera [0 100] - Topologia Mestre-trabalhador

5.5.2 Topologia em Anel

Nesta seção iremos implementar a estratégia paralela para comunicação em Anel. Iniciamos as experimentações, com 4 processadores, de forma a caracterizar a estrutura de troca

de mensagens em forma de anel. Utilizando inicialmente 4 PEs, cada um terá um vizinho do lado direito e esquerdo para troca informações. Na sequência aumentamos este número para 8, 10 e 12 PEs. Ao término das experimentações faremos uma análise comparativa conforme o aumento no número de processadores. Como em nossos experimentos cada processador só trabalha com as partículas dentro de seu espaço de busca, as trocas de mensagens que ocorrem entre os PEs, são utilizadas para identificar os valores de *pbest* dos demais PEs. De posse destes valores cada PE identifica se deve ou não continuar oo processo de otimização dentro de seu espaço.

5.5.2.1 Topologia em Anel utilizando 4 processadores

Os resultados experimentais obtidos quando utilizamos 4 processadores são apresentados na Tabela 23. A penúltima linha apresenta as médias considerando o efeito estocástico de 10 simulações. Encontramos a duração de 10,75 ms e 86 iterações quando utilizamos a função Rosenbrock. Para a função Esfera obtivemos a duração de 7,44 ms e 60 iterações para o espaço de busca [0,50] e a duração de 57,25 ms e 347 iterações para o espaço de busca [0,100]. Os valores obtidos para o *speedup* estão na última linha da tabela e assim como na estratégia Mestre-trabalhador encontramos valores superlineares devido a diminuição da quantidade de partículas quando dividimos o enxame entre os PEs, reduzindo assim o esforço computacional necessário ao cálculo do PSO. A alocação dos processadores no ambiente de simulação ficou conforme indicado na Figura 34, onde podemos observar através da indicação das setas que a troca de mensagens ocorre em forma circular caracterizando a topologia em anel. Conforme fizemos na topologia Mestre-trabalhador, dividimos a tarefa principal de acordo com o número de processadores. Neste caso, que temos 4 PEs, dividimos em outras quatro tarefas de forma a executá-las simultaneamente. As quatro tarefas estão identificadas na figura por PSO A1 até PSO A4.

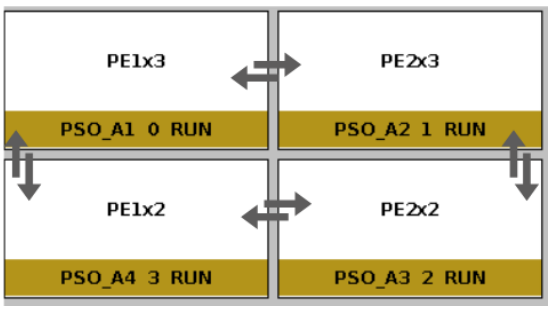


Figura 34: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Anel utilizando 4PEs

Tabela 23: Número de iterações e tempo de execução da versão paralela do PSO com topologia Anel utilizando 4 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	137	17,26	4	0,73	208	34,39
Simulação 2	46	6,05	119	14,67	1	0,56
Simulação 3	22	3,01	43	5,50	1	0,95
Simulação 4	188	23,32	104	12,85	80	14,01
Simulação 5	20	2,64	65	8,39	2042	332,707
Simulação 6	104	12,95	27	3,42	326	54,18
Simulação 7	105	12,91	110	13,46	138	23,87
Simulação 8	73	9,14	1	0,29	286	47,73
Simulação 9	107	13,34	34	4,24	119	19,50
Simulação 10	53	6,84	25	3,18	267	44,62
Média	86	10,75	60	7,44	347	57,25
speedup	18,21		11,73		3,31	

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 24, sendo de 6,18 ms, 5,07 ms e 93,51 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 24: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Anel utilizando 4 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	10,75	7,44	57,25
Desvio padrão (ms)	6,18	5,07	93,51

Os tempos referentes à troca de mensagens foram calculados a partir da Tabela 25. Para a função Rosenbrock o tempo médio de troca de mensagens representa 43,77% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 35,87% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 27,54%.

Tabela 25: Tempo de troca de mensagens da versão paralela do PSO com topologia Anel utilizando 4 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	7,62	0,13	11,66
Simulação 2	2,55	6,42	0
Simulação 3	1,21	2,27	0
Simulação 4	10,44	5,57	4,56
Simulação 5	0,99	3,41	112,99
Simulação 6	5,92	1,43	18,45
Simulação 7	5,92	6,18	8,48
Simulação 8	4,17	0	16,21
Simulação 9	6,16	1,76	6,95
Simulação 10	3,08	1,32	16,01
Média	4,81	3,19	19,53

5.5.2.2 Topologia em Anel utilizando 8 processadores

Os resultados obtidos quando utilizamos estão apresentados na Tabela 26. Encontramos a duração média T_S de 5,79 ms e 41 iterações quando utilizamos a função Rosenbrock. Para a função Esfera obtivemos a duração de 3,90 ms e 23 iterações para o espaço de busca [0,50] e a duração de 4,65 ms e 33 iterações para o espaço de busca [0,100].

Utilizando 8 PEs a alocação no ambiente de simulação MENPHIS, ficou conforme indicado na Figura 35, que mostra também o nome das 8 tarefas e os sentidos dos fluxo de troca de mensagens.

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 27, sendo de 5,42 ms, 4,08 ms e 5,43 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Os dados referentes a troca de mensagens foram calculados a partir das informações da Tabel 28. Para a função Rosenbrock o tempo médio de troca de mensagens representa 39,02% do tempo total do experimento, para a função Esfera no espaço de busca [0,50]

Tabela 26: Número de iterações e tempo de execução da versão paralela do PSO com topologia Anel utilizando 8 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	8	1,37	1	0,28	4	0,89
Simulação 2	20	3,45	10	1,93	3	0,80
Simulação 3	40	6,26	9	1,55	3	0,93
Simulação 4	21	3,46	1	0,38	4	1,17
Simulação 5	2	0,48	45	6,94	137	19,31
Simulação 6	4	0,87	11	2,27	15	2,16
Simulação 7	55	7,44	4	0,87	58	7,58
Simulação 8	112	14,87	68	11,42	54	7,14
Simulação 9	126	16,67	66	11,09	25	3,46
Simulação 10	21	3,07	11	2,24	21	3,03
Média	41	5,79	23	3,90	33	4,65
speedup	33,78		22,38		40,82	

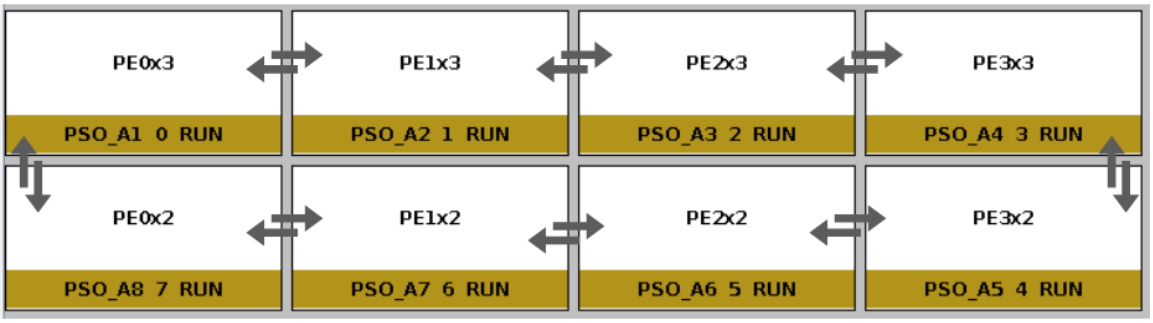


Figura 35: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Anel utilizando 8PEs

Tabela 27: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Anel utilizando 8 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	5,79	3,90	4,65
Desvio padrão (ms)	5,42	4,08	5,43

este valor ficou em 36,78% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 32,66%.

Tabela 28: Tempo de troca de mensagens da versão paralela do PSO com topologia Anel utilizando 8 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	0,50	0	0,12
Simulação 2	1,63	0,84	0,19
Simulação 3	3,32	0,69	0,23
Simulação 4	1,71	0	0,33
Simulação 5	0,07	3,75	7,94
Simulação 6	0,19	0,98	0,73
Simulação 7	3,18	0,19	3,11
Simulação 8	6,47	6,03	3,01
Simulação 9	7,26	6,06	1,37
Simulação 10	1,12	1,14	1,10
Média	2,54	1,99	1,81

5.5.2.3 Topologia em Anel utilizando 10 processadores

Os resultados experimentais obtidos quando utilizamos 10 processadores são apresentados na Tabela 29. Encontramos a duração de 0,98 ms e 5 iterações quando utilizamos a função Rosenbrock. Para a função Esfera obtivemos a duração de 1,52 ms e 11 iterações para o espaço de busca [0,50] e a duração de 5,02 ms e 43 iterações para o espaço de busca [0,100]. A alocação dos processadores no ambiente de simulação ficou conforme indicado na Figura 36 .

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 30, sendo de 1,87 ms, 2,69 ms e 3,71 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Os dados referentes a troca de mensagens foram calculados a partir das informações da Tabela 31. Para a função Rosenbrock o tempo médio de troca de mensagens representa 16,49% do tempo total do experimento, para a função Esfera no espaço de busca [0,50]

Tabela 29: Número de iterações e tempo de execução da versão paralela do PSO com topologia Anel utilizando 10 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	38	6,59	1	0,41	43	6,53
Simulação 2	1	0,33	1	0,38	5	1,06
Simulação 3	1	0,40	63	9,37	30	4,59
Simulação 4	1	0,42	1	0,31	37	5,53
Simulação 5	3	0,76	6	0,94	34	4,87
Simulação 6	1	0,25	2	0,34	4	0,59
Simulação 7	1	0,23	2	0,34	115	9,76
Simulação 8	1	0,26	1	0,22	30	4,74
Simulação 9	1	0,29	28	2,55	39	5,50
Simulação 10	1	0,26	2	0,38	84	7,052
Média	5	0,98	11	1,52	43	5,02
speedup	198,80		57,17		37,78	

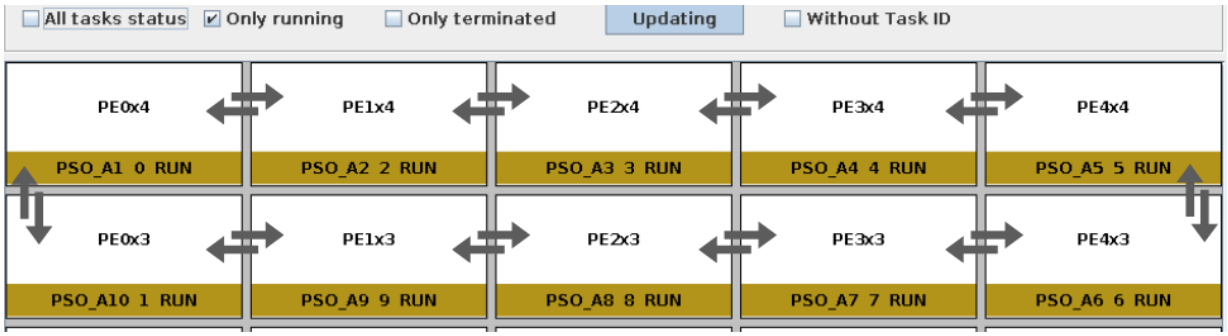


Figura 36: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Anel utilizando 10PEs

Tabela 30: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Anel utilizando 10 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	0,98	1,52	5,02
Desvio padrão (ms)	1,87	2,69	3,71

este valor ficou em 19,43% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 34,93%.

Tabela 31: Tempo de troca de mensagens da versão paralela do PSO com topologia Anel utilizando 10 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	2,94	0	3,34
Simulação 2	0	0	0,28
Simulação 3	0	5	2,31
Simulação 4	0	0	2,85
Simulação 5	0,15	0,33	2,37
Simulação 6	0	0,09	0,08
Simulação 7	0	0,08	3,28
Simulação 8	0	0	0,85
Simulação 9	0	0,85	0,85
Simulação 10	0	0,07	3,87
Média	0,30	0,64	2,03

5.5.2.4 Topologia em Anel utilizando 12 processadores

Os resultados médios obtidos quando utilizamos 12 processadores são apresentados na Tabela 32. Encontramos a duração de 26,90 ms e 192 iterações quando utilizamos a função Rosenbrock. Para a função Esfera obtivemos a duração de 20,46 ms e 148 iterações para o espaço de busca [0,50] e a duração de 38,43 ms e 278 iterações para o espaço de busca [0,100]. Os valores de *speedup* estão ma última linha da tabela. A alocação dos processadores no ambiente de simulação ficou conforme indicado na Figura 37, com as respectivas indicações das 12 tarefas e do fluxo referente a troca de mensagens.

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 33, sendo de 34,74 ms, 16,38 ms e 32,01 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Os valores de tempo médio gasto com trocas de mensagem estão na Tabela 34. Para a função Rosenbrock corresponde a 53,32% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 46,82% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 49,59%.

Tabela 32: Número de iterações e tempo de execução da versão paralela do PSO com topologia Anel utilizando 12 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	376	52,75	360	49,89	60	8,33
Simulação 2	58	8,17	1	0,37	67	9,31
Simulação 3	88	12,44	101	14,06	2	0,46
Simulação 4	1	0,35	69	9,55	264	36,60
Simulação 5	533	75,04	241	33,30	368	50,94
Simulação 6	730	102,52	76	10,53	477	66,09
Simulação 7	2	0,50	103	14,23	207	28,84
Simulação 8	40	5,56	325	45,02	84	11,70
Simulação 9	1	0,26	17	2,36	478	66,12
Simulação 10	82	11,41	183	25,30	766	105,92
Média	192	26,90	148	20,46	278	38,43
speedup	7,28		4,27		4,94	

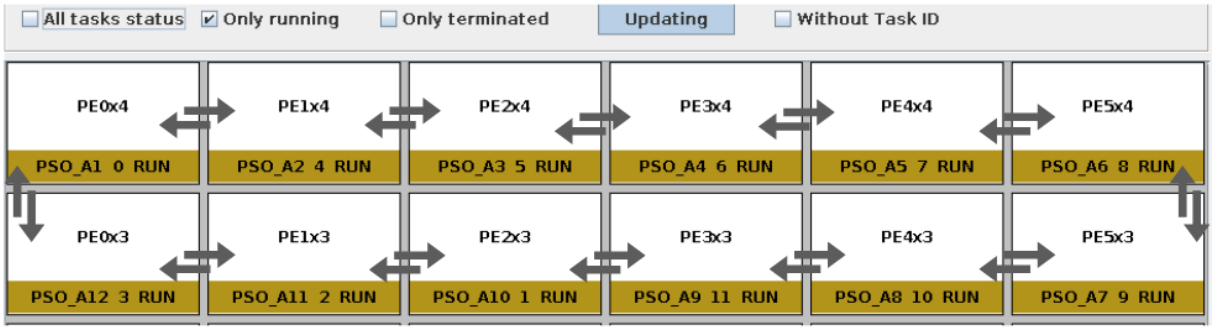


Figura 37: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Anel utilizando 12PEs

Tabela 33: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Anel utilizando 12 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	26,90	20,46	38,43
Desvio padrão (ms)	34,74	16,38	32,01

5.5.2.5 Análise comparativa *speedup* para estratégia em Anel

Apresentaremos aqui um comparativo gráfico quando utilizamos a estratégia paralela com a topologia em Anel variando o número de processadores. Os gráficos obtidos estão apre-

Tabela 34: Tempo de troca de mensagens da versão paralela do PSO com topologia Anel utilizando 12 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	28,42	26,73	4,35
Simulação 2	4,22	0	4,86
Simulação 3	6,54	7,37	0,08
Simulação 4	0	4,93	19,48
Simulação 5	40,50	17,77	27,22
Simulação 6	55,43	5,50	35,45
Simulação 7	0,83	7,48	15,49
Simulação 8	2,74	24,21	6,13
Simulação 9	0	1,06	35,58
Simulação 10	5,92	13,51	57,07
Média	14,46	10,86	20,57

sentados a partir da Figura 38. Apresentaremos aqui um comparativo gráfico quando utilizamos a estratégia paralela com a topologia em Anel variando o número de processadores. Para a função Rosenbrock o gráfico obtido está Figura 38. Para a função Esfera no intervalo [0,50] e Esfera [0,100] os gráficos estão respectivamente nas Figuras 39 e 40.

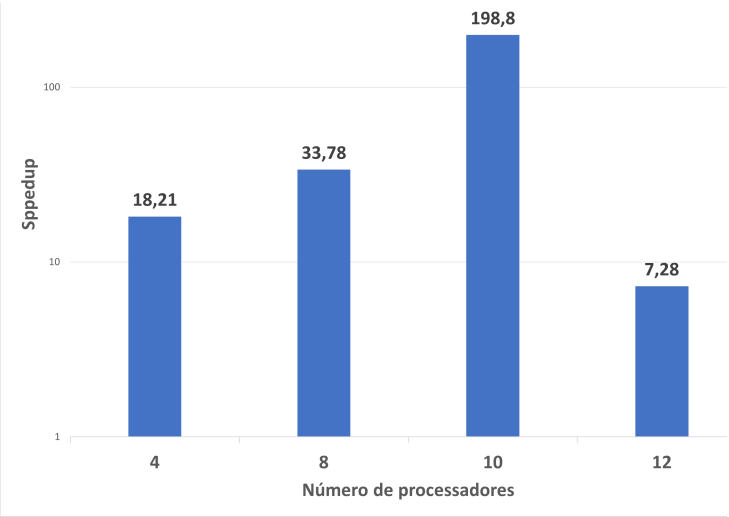


Figura 38: Speedup x número de processadores Função Rosenbrock - Topologia Anel

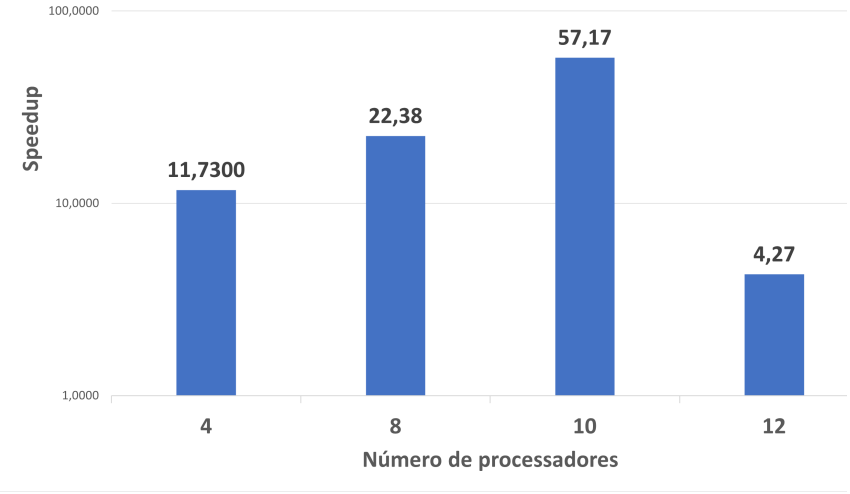


Figura 39: Speedup x número de processadores Função Esfera [0,50] - Topologia Anel

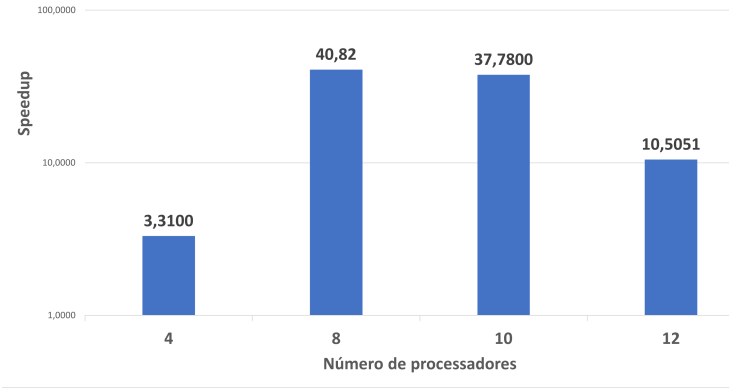


Figura 40: Speedup x número de processadores Função Esfera [0,100] - Topologia Anel

5.5.3 Topologia em Malha 2D

A característica mais importante desta estratégia é a possibilidade de comunicação com o número máximo de 4 processadores da vizinhança na Malha 2D, ou seja, cima, baixo, direita e esquerda. A escolha dos locais de mapeamento dos processadores na plataforma de experimentação, visou propiciar a maior quantidade de processadores vizinhos para troca de mensagens.

5.5.3.1 Topologia Malha 2D utilizando 8 processadores.

Os dados obtidos são apresentados na Tabela 35.

Quando utilizamos a função Rosenbrock encontramos a duração de 5,22 ms e 27 iterações. Para a função Esfera obtivemos a duração de 4,67 ms e 24 iterações para o espaço de busca [0,50] e a duração de 11,34 ms e 62 iterações para o espaço de busca [0,100].

Tabela 35: Número de iterações e tempo de execução da versão paralela do PSO com topologia Malha 2D utilizando 8 PEs.

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	9	2,08	15	3,02	23	3,56
Simulação 2	64	12,40	5	1,04	39	7,81
Simulação 3	1	0,53	88	16,60	78	14,43
Simulação 4	43	8,55	58	11,14	5	0,97
Simulação 5	6	1,45	5	1,45	5	1,10
Simulação 6	51	9,88	26	5,05	156	28,87
Simulação 7	27	5,28	28	5,34	45	7,08
Simulação 8	6	1,33	1	0,49	16	3,27
Simulação 9	1	0,48	7	1,47	109	20,35
Simulação 10	53	10,19	5	1,11	139	25,96
Média	27	5,22	24	4,67	62	11,34
speedup	37,50		18,68		16,74	

A alocação dos processadores, divisão das tarefas e indicação de trocas de mensagens no ambiente de simulação, ficou conforme indicado na Figura 41 .

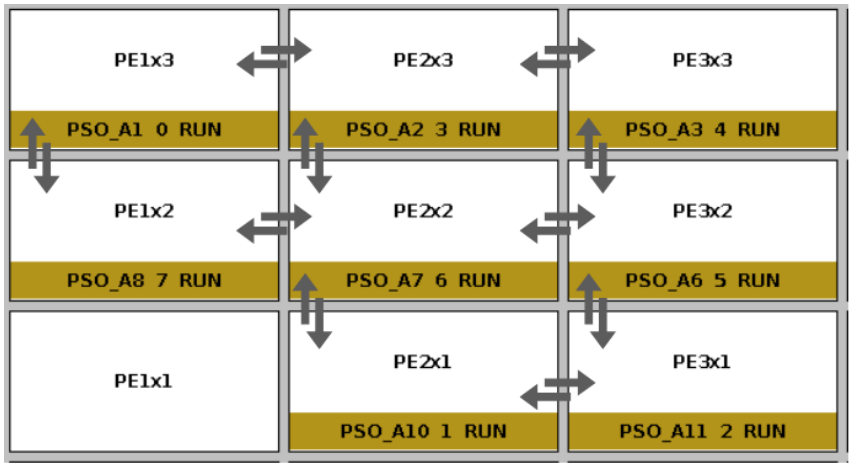


Figura 41: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Malha 2D utilizando 8PEs.

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 36, sendo de 4,38 ms, 5,01 ms e 9,90 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Os valores de tempo médio gasto com trocas de mensagem estão na Tabela 37. Para a função Rosenbrock corresponde a 32,59% do tempo total do experimento, para a

Tabela 36: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Malha 2D utilizando 8 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	5,22	4,67	11,34
Desvio padrão (ms)	4,38	5,01	9,90

função Esfera no espaço de busca [0,50] este valor ficou em 37,51% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 57,17%.

Tabela 37: Tempo de troca de mensagens da versão paralela do PSO com topologia Malha 2D utilizando 8 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	0,91	1,61	1,85
Simulação 2	6,92	0,35	4,47
Simulação 3	0	9,43	8,25
Simulação 4	4,60	6,25	0,25
Simulação 5	0,52	0,52	0,38
Simulação 6	5,43	2,69	16,50
Simulação 7	2,46	2,75	4,70
Simulação 8	0,45	0	1,36
Simulação 9	0	0,50	11,72
Simulação 10	5,64	0,37	14,93
Média	2,69	2,45	6,44

5.5.3.2 Topologia Malha 2D utilizando 10 processadores

Conforme a Tabela 38, encontramos a duração de 3,47 ms e 19 iterações para a função Rosenbrock, 0,80 ms e 3 iterações para a função Esfera no espaço de busca [0,50] e 4,54 ms e 39 iterações para o espaço de busca [0,100].

Tabela 38: Número de iterações e tempo de execução da versão paralela do PSO com topologia Malha 2D utilizando 10 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	59	9,87	2	0,52	83	13,73
Simulação 2	3	0,74	1	0,52	943	155,99
Simulação 3	39	6,65	4	0,94	5	1,09
Simulação 4	40	7,03	4	0,99	4	1,01
Simulação 5	20	4,31	1	0,66	4	1,01
Simulação 6	1	0,47	1	0,36	86	7,46
Simulação 7	1	0,41	1	0,38	4	0,57
Simulação 8	1	0,35	9	1,80	1	0,31
Simulação 9	1	0,38	3	0,83	97	8,38
Simulação 10	25	4,48	4	1	57	5,17
Média	19	3,47	3	0,80	39	4,54
speedup	56,41		108,62		41,74	

Os valores de *speedup* estão na última linha da tabela. A alocação dos processadores no ambiente de simulação ficou conforme indicado na Figura 42, com as respectivas indicações das 10 tarefas e do fluxo referente a troca de mensagens.

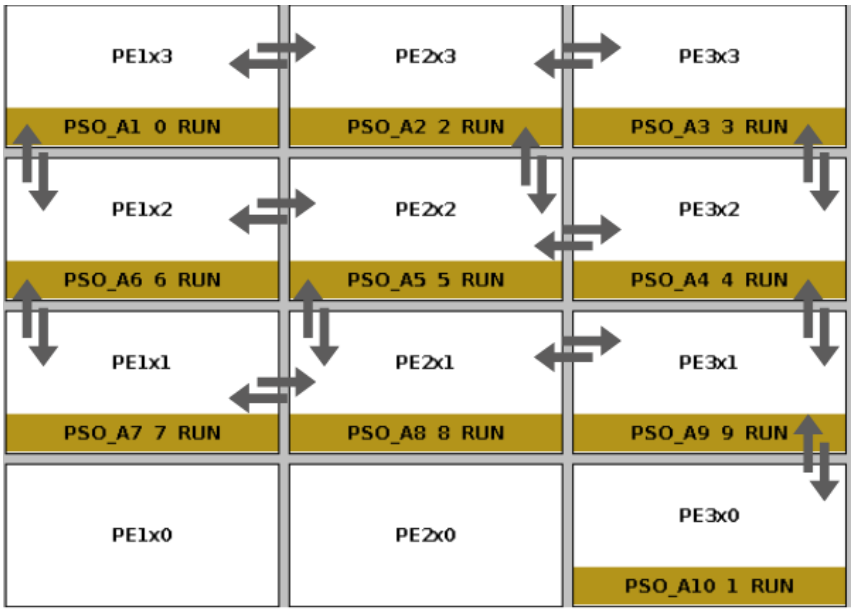


Figura 42: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Malha 2D utilizando 10PEs

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 39, sendo de 3,32 ms , 0,4 ms e 4,33 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 39: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Malha 2d utilizando 10 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	3,47	0,80	4,54
Desvio padrão (ms)	3,32	0,4	4,33

Os valores de tempo médio gasto com trocas de mensagem estão na Tabela 40. Para a função Rosenbrock corresponde a 28,32% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 21,40% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 29,74%.

Tabela 40: Tempo de troca de mensagens da versão paralela do PSO com topologia Malha 2D utilizando 10 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	5,25	0,09	7,44
Simulação 2	0,17	0	0,10
Simulação 3	3,54	0,31	0,39
Simulação 4	3,70	0,32	3,70
Simulação 5	2,16	0	0,37
Simulação 6	0	0	2,33
Simulação 7	0	0	0,07
Simulação 8	0	0,88	0
Simulação 9	0	0,31	2,48
Simulação 10	2,24	0,41	1,51
Média	1,71	0,23	1,84

5.5.3.3 Topologia Malha 2D utilizando 12 processadores

Para a simulação utilizando 12 PEs, os dados obtidos são apresentados na Tabela 41.

Tabela 41: Número de iterações e tempo de execução da versão paralela do PSO com topologia Malha 2D utilizando 12 PEs

	Rosenbrock [0,50]		Esfera [0,50]		Esfera [0,100]	
	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)	N. iterações	Tempo de execução (ms)
Simulação 1	2	0,47	5	1,22	118	19,62
Simulação 2	377	63,58	44	7,36	165	27,46
Simulação 3	37	6,41	3	0,60	205	34,22
Simulação 4	3	0,73	50	8,48	128	21,39
Simulação 5	5	1,07	88	14,81	174	28,63
Simulação 6	3	0,80	113	18,99	80	13,50
Simulação 7	273	46,18	5	1,06	153	25,64
Simulação 8	98	16,78	76	12,86	230	38,32
Simulação 9	134	22,83	81	13,66	53	9
Simulação 10	26	4,61	51	8,69	285	47,43
Média	96	16,35	52	8,77	160	26,52
speedup	11,97		9,95		7,15	

Quando utilizamos a função Rosenbrock encontramos a duração de 16,35 ms e 96 iterações. Para a função Esfera obtivemos a duração de 8,77 ms e 52 iterações para o espaço de busca [0,50] e a duração de 26,52 ms e 160 iterações para o espaço de busca [0,100]. Os valores de *speedup* estão na última linha da tabela. A alocação dos processadores no ambiente de simulação ficou conforme indicado na Figura 43, com as respectivas indicações das 12 tarefas e do fluxo referente a troca de mensagens.

Para o cálculo do desvio padrão das durações referentes as simulações, obtivemos os resultados conforme a Tabela 42, sendo de 20,9 ms, 6,05 ms e 10,91 ms respectivamente para as funções Rosenbrock [0,50], Esfera [0,50] e Esfera [0,100].

Tabela 42: Valores de duração média e desvio padrão para versão paralela do PSO com topologia Malha 2d utilizando 12 PEs

	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Média (ms)	16,35	8,77	26,52
Desvio padrão (ms)	20,9	6,05	10,91

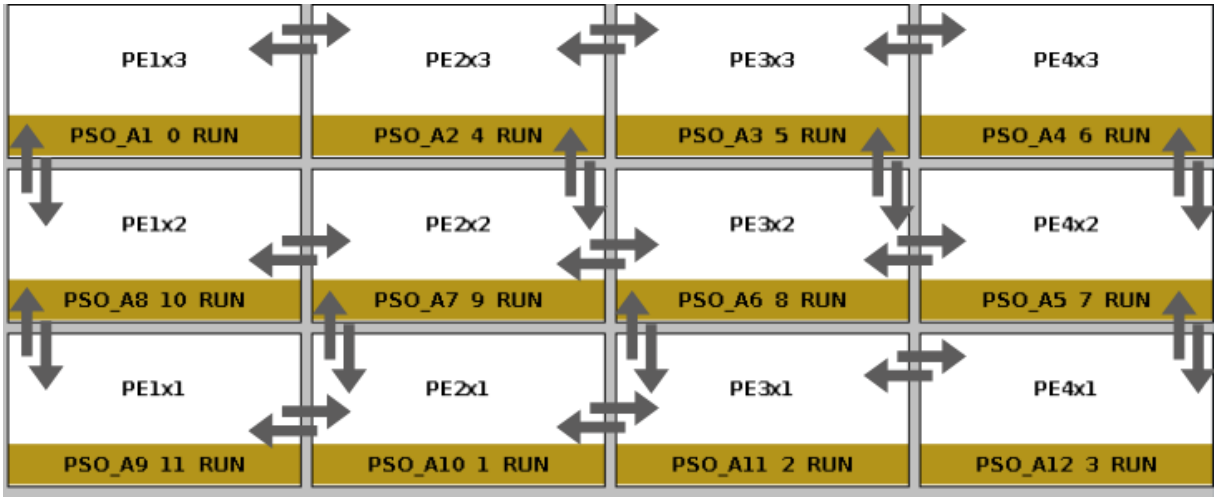


Figura 43: Mapeamento de tarefas na MEMPHIS para a versão paralela do PSO com topologia Malha 2d utilizando 12PEs

Os valores de tempo médio gasto com trocas de mensagem estão na Tabela 43. Para a função Rosenbrock corresponde a 28,32% do tempo total do experimento, para a função Esfera no espaço de busca [0,50] este valor ficou em 21,40% e finalmente para a função Esfera no espaço de busca [0,100] este valor ficou em 29,74%.

Tabela 43: Tempo de troca de mensagens da versão paralela do PSO com topologia Anel utilizando 12 PEs

	Tempo de troca de mensagens (ms)		
	Rosenbrock [0,50]	Esfera [0,50]	Esfera [0,100]
Simulação 1	0,11	0,42	11,66
Simulação 2	38,18	4,25	16,34
Simulação 3	3,64	0,13	20,44
Simulação 4	0,21	4,87	12,89
Simulação 5	0,33	8,78	17,29
Simulação 6	0,31	11,33	8,03
Simulação 7	27,86	0,45	15,46
Simulação 8	9,98	7,57	23,07
Simulação 9	13,66	8,05	5,26
Simulação 10	2,63	5,13	28,55
Média	9,69	5,10	15,90

5.5.3.4 Análise comparativa *speedup* para estratégia em Malha 2D

Nesta seção faremos o comparativo gráfico ao utilizarmos a estratégia paralela com a topologia em Malha 2D variando o número de processadores. Os gráfico obtidos quando utilizamos a função Rosenbrock está apresentados na Figura 44, para a função Esfera no intervalo [0,50] na Figura 45 e para a função Esfera no intervalo [0,100] na Figura 46

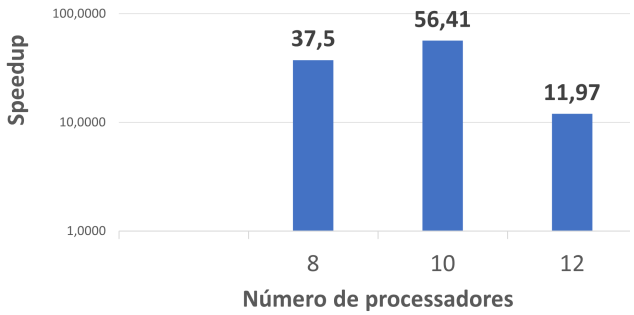


Figura 44: Speedup x número de processadores Função Rosenbrock - Topologia Malha 2D

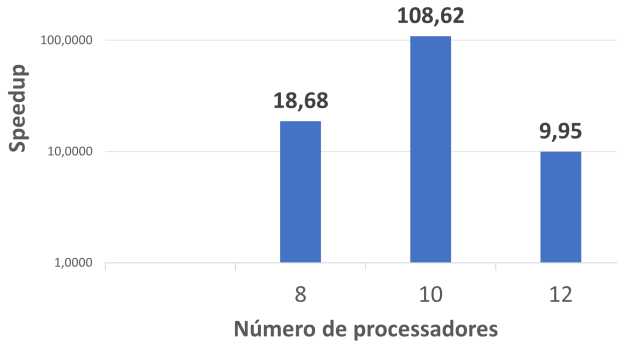


Figura 45: Speedup x número de processadores Função Esfera [050] - Topologia Malha 2D

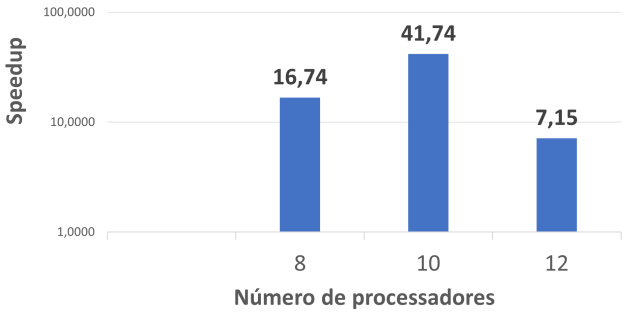


Figura 46: Speedup x número de processadores Função Esfera [0 100] - Topologia Malha 2D

5.6 Considerações finais

Neste capítulo foram apresentados os resultados para as simulações realizadas para o PSO em sua forma serial e utilizando estratégias paralelas. Foram utilizados 3 topologias diferentes Mestre-trabalhador, Anel e Malha 2D com número de processadores variando de 2 até 12. Pode-se observar em uma rápida análise, que a função Rosenbrock teve uma melhor desempenho, quando comparamos com a função Esfera, ao utilizar o paralelismo. Tal fato se deve ao fato de ser mais complexa que a função Esfera para ser otimizada. Como a versão serial demora significativamente para encontrar uma primeira partícula que encontre o valor de mínimo, dividir o espaço de busca e fazer a busca utilizando vários processadores se apresentou bastante vantajoso. Quanto ao tempo gasto em troca de mensagens, a estratégia Mestre-trabalhador a foi a que demandou mais tempo. A atualização, dos processadores trabalhadores, a cada iteração onerou bastante este tempo. As estratégias utilizando topologias em Anel e Malha 2D obtiveram um menor tempo de troca de mensagens e um nível ótimo de *speedup*, apesar de inferior ao obtido na estratégia Mestre-trabalhador. Importante também observar que como cada PE trabalha apenas em seu espaço de busca e utiliza a informação dos PEs vizinhos apenas para verificar se continua ou não o processo de otimização, os valores de *speedup* obtidos podem variar devido ao fator estocástico inerente ao cálculo do PSO.

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

NESTE capítulo, apresentamos as principais conclusões tiradas a partir da observação dos resultados obtidos. São apresentadas conclusões sobre os algoritmos e topologias propostas bem como pontos a serem explorados em trabalhos futuros envolvendo o paralelismo. A Seção 6.1 destaca as principais conclusões referentes a esta dissertação. Enquanto a Seção 6.2 explora sugestões para trabalhos futuros.

6.1 Conclusões

O objetivo deste trabalho foi obter uma implementação paralela do PSO em um sistema embutido multiprocessado com rede intrachip e analisar o *speedup* que essa implementação poderia proporcionar. Para isso, utilizamos uma plataforma de simulação chamada MEMPHIS, que fornece a possibilidade de determinar o número de processadores (PEs) a serem utilizados, retornando o tempo de execução e o tempo de troca de mensagem entre tarefas. Para esses experimentos, utilizamos até 12 processadores e o espaço de busca foi dividido entre eles, a fim de localizar o mínimo das funções Rosenbrock e Esfera. Aplicamos três tipos de topologias: Mestre-trabalhador, Anel e Malha 2D. Os resultados obtidos são muito promissores.

A escolha do PSO se deu devido à sua utilização em larga escala na resolução de problemas de otimização dos mais variados tipos de problemas em diferentes áreas. A principal ideia ao se dividir o espaço de busca foi aproveitar a disponibilidade de mais PEs, para criar tarefas, que podem ser executadas simultaneamente e com o mesmo objetivo de localizar o mínimo das funções. Um ponto chave, que foi considerado, é referente ao tempo de troca de mensagem gasto entre os processadores conforme os diferentes tipos

de topologias utilizadas. O objetivo é que mesmo com este tempo gasto em troca de mensagem o paralelismo traga vantagem sobre o modelo serial.

A função Rosenbrock no espaço de busca [0 50], devido às características de sua curva, que tem muitos mínimos locais e uma única solução global mínima, apresenta uma maior dificuldade durante o processo de otimização. Esta complexidade gera uma demora considerável na execução do PSO na versão serial e por isso foi a mais beneficiada quanto a valores de *speedup* ao utilizarmos vários processadores. O melhor *speedup* foi de 315,78 com 10 PEs e um valor médio de 2 iterações na topologia Mestre-trabalhador. Encontramos valores *speedup* de 198,8 com 10 PEs e média de 5 iterações e 56,41 com 10 PEs e média de 19 iterações quando utilizamos as topologias em Anel e Malha 2D respectivamente.

Para a função Esfera, no espaço de busca [0,50], o melhor *speedup* foi de 154,01 com 10 PEs e média de 3 iterações na topologia Mestre-trabalhador, enquanto as topologias em Anel e Malha 2D ficaram respectivamente com valores de *speedup* de 57,17 com 10 PEs utilizando uma média de 3 iterações e 108,62 e 10 PEs com média de 11 iterações. Já para a mesma função, no espaço de busca [0,100], a topologia Malha 2D apresentou o melhor valor de *speedup* de 41,74 com 10 PEs e média de 39 iterações enquanto a topologia em Anel 40,82 com 8 PEs e média de 33 iterações e a Mestre-trabalhador 19,57 e média de 82 iterações. Vale ressaltar que este aumento no espaço de busca dificultou bastante a tarefa sendo este o valor máximo de *speedup* encontrado.

Os melhores valores de *speedup* são obtidos quando utilizamos 8 ou 10 PEs. Tal fato ocorre em função da dinâmica utilizada de divisão do espaço de busca e também do número total de partículas entre os PEs que atinge um ponto aproximado de equilíbrio com esta quantidade de PEs. Quando utilizamos 12 PEs, este equilíbrio é quebrado, pois apesar da diminuição no espaço de busca associado a cada PE, o número de partículas também diminui sendo insuficiente para uma otimização adequada, ocasionando um valor de *speedup* significativamente inferior ao que obtivemos quando utilizamos um número inferior de PEs. De forma geral, a divisão do espaço de busca em áreas menores, distribuídas entre os processadores, ajudou significativamente a reduzir o esforço computacional requerido por cada um.

Quanto ao tempo de troca de mensagens, fizemos o comparativo percentual em relação ao tempo total de simulação. Quando utilizamos a topologia Mestre-trabalhador

e a função Rosenbrock, este percentual variou entre 12,24%, quanto utilizamos 2 processadores, e 78,74% para 12 PEs. Quando comparamos este resultado com os obtidos pela topologia em Anel e Malha 2D, podemos constatar que conforme aumentamos o número de PEs, ocorre uma diminuição significativa do percentual de troca de mensagens. A estratégia Mestre-trabalhador fica bastante onerada quando avaliamos o tempo de troca de mensagens entre os processadores, devido à atualização de melhor local a cada iteração para todas as partículas. Utilizando a estratégia em Anel obtivemos um percentual máximo de troca de mensagens de 42,4227% e 46,1532% quanto utilizamos Malha 2D, considerando 12 PEs.

Para a função Esfera [0,50] utilizando a topologia Mestre-trabalhador, o percentual gasto em tempo de troca de mensagens em comparação com o tempo total de execução, também foi expressivo e variou entre 16,73%, quando utilizamos 2 processadores, e 96,94% para 12 PEs. Já para a topologia em Anel e Malha 2D os percentuais de comunicações quando utilizamos 12 PEs foram respectivamente de 49,59% e 59,80% , que também são memores quando comparados com o tempo de troca de mensagens gasto quando utilizamos a estratégia Mestre-trabalhador.

Finalmente, quando consideramos a função Esfera no intervalo [0,100], os valores percentuais representativos do tempo de troca de mensagens também foram semelhantes quando fazemos o comparativos entre as topologias utilizadas. Os valores percentuais de troca de mensagens ficaram respectivamente entre 12,51% e 85,32% quando utilizamos a topologia Mestre-trabalhador variando o número de processadores entre 2 e 12. As topologias Anel e Malha 2D obtiveram um percentual de troca de mensagens respectivamente entre 49,59% e 59,80% quando utilizamos o número máximo de 12 PEs.

A ideia principal deste trabalho foi mostrar a possibilidade de explorar paralelismo na execução do algoritmo PSO, utilizando uma plataforma multiprocessada, como a MEMPHIS. Foi possível dividir o processo de execução do PSO, entre os processadores de forma escalável e obter valores de *speedup* significativos. A topologia Mestre-trabalhador obteve bons resultados porém, um alto volume de troca de mensagens, enquanto as topologias em Anel e Malha 2D um bom desempenho com um volume de troca de mensagens relativamente menor.

6.2 Trabalhos Futuros

A MEMPHIS apresenta algumas limitações, como ausência de suporte a funções trigonométricas e operações de ponto flutuante. Estes fatores são necessários para conseguirmos uma assertividade e refinamento dos dados obtidos.

Outra ponto a ser explorado é quanto ao uso de outras topologias, para troca de comunicação, além das utilizadas neste trabalho, de forma a identificar qual apresenta menor impacto no tempo total de processamento. Pudemos verificar neste trabalho que a topologia Mestre-trabalhador onera significativamente o tempo de processamento. Alterações ou uso de outros algoritmos diferentes do Local Best PSO, Global Best PSO e Cooperative PSO utilizados em nossas simulações bem como o uso de estratégias que utilizem forma assíncrona para atualização das partículas também, podem ser explorados.

Neste trabalho cada PE ficou limitado a seu espaço de busca, utilizando a informação dos vizinhos apenas para validar a continuidade de otimização em seu espaço. Outra frente é a possibilidade que temos para configurar todo espaço de busca em todos os processadores. Utilizando esta estratégia, será possível identificar o grau de convergência de todo o enxame e comparar com a solução quando apenas o subenxame, localizado no espaço de busca que contém a solução, converge para o valor de mínimo da função.

REFERÊNCIAS

ALTINOZ, O. T. et al. Particle swarm optimization with social exclusion and its application in electromagnetics. In: IEEE. *2014 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*. [S.l.], 2014. p. 105–110.

ARBOLEDA, D. M. M. et al. Hardware architecture for particle swarm optimization using floating-point arithmetic. In: *2009 Ninth International Conference on Intelligent Systems Design and Applications*. [S.l.: s.n.], 2009. p. 243–248.

ASHCROFT, E. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, v. 10, n. 1, p. 110–135, 1975. ISSN 0022-0000. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0022000075800183>.

BENINI, L.; MICHELI, G. D. Networks on chip: A new paradigm for systems on chip design. In: IEEE. *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. [S.l.], 2002. p. 418–419.

BERGH, F. Van den; ENGELBRECHT, A. P. A cooperative approach to particle swarm optimization. *IEEE transactions on evolutionary computation*, IEEE, v. 8, n. 3, p. 225–239, 2004.

BURIOL, T. M.; ARGENTA, M. A. Acelerando o desenvolvimento e o processamento de análises numéricas computacionais utilizando python e cuda. *Métodos Numéricos e Computacionais em Engenharia-CMNE CILAMCE*, 2009.

CALAZAN, R. d. M. et al. Otimização por enxame de partículas em arquiteturas paralelas de alto desempenho. Universidade do Estado do Rio de Janeiro, 2013.

CARARA, E.; CALAZANS, N.; MORAES, F. A new router architecture for high-performance intrachip networks. *Journal of Integrated Circuits and Systems*, v. 31, p. 23–31, 11 2008.

CHAPMAN, B.; JOST, G.; PAS, R. V. D. *Using OpenMP*. [S.l.]: The MIT Press Cambridge, 2008.

CLERC, M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: IEEE. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. [S.l.], 1999. v. 3, p. 1951–1957.

CLERC, M. *Particle swarm optimization*. [S.l.]: John Wiley & Sons, 2010.

CORMEN, T. et al. *Advanced Algorithms-CS 6/76101*. [S.l.]: Citeseer, 2001.

COSTA, A. L. da et al. Proposta de implementação paralela do algoritmo pso para fpga. In: *Congresso Brasileiro de Automática-CBA*. [S.l.: s.n.], 2019. v. 1, n. 1.

DAMAJ, I. et al. An analytical framework for high-speed hardware particle swarm optimization. *Microprocessors and Microsystems*, v. 72, p. 102949, 2020. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933119300407>.

DIETRICH, R.; JUCKELAND, G.; WOLFE, M. Open acc programs examined: a performance analysis approach. In: IEEE. *2015 44th International Conference on Parallel Processing*. [S.l.], 2015. p. 310–319.

ENGELBRECHT, A. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005. ISBN 9780470091913. Disponível em: <https://books.google.com.br/books?id=3xhrQgAACAAJ>.

ENGELBRECHT, A. P. *Fundamentals of computational swarm intelligence*. [S.l.]: John Wiley & Sons, Inc., 2006.

FARBER, R. *Parallel programming with OpenACC*. [S.l.]: Newnes, 2016.

FARMAHINI-FARAHANI, A. et al. Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. *Engineering Applications of Artificial Intelligence*, v. 23, n. 2, p. 177–187, 2010. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0952197609001584>.

FOSTER, I. *Designing and building parallel programs: concepts and tools for parallel software engineering*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1995.

GHEIN, L. D.; FUNDAMENTALS, M. *Cisco Press*. [S.l.]: November, 2006.

GÜLCÜ, Ş.; KODAZ, H. A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 45, p. 33–45, 2015.

HUNG, W. W. Y. Accelerating parallel particle swarm optimization via gpu. *Optimization Methods and Software*, Taylor Francis, v. 27, n. 1, p. 33–51, 2012. Disponível em: <https://doi.org/10.1080/10556788.2010.509435>.

KARINIEMI, H.; NURMI, J. Arbitration and routing schemes for on-chip packet networks. In: *Interconnect-centric design for advanced SoC and NoC*. [S.l.]: Springer, 2004. p. 253–282.

KENNEDY, J. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: IEEE. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. [S.l.], 1999. v. 3, p. 1931–1938.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. *Proceedings of ICNN'95-international conference on neural networks*. [S.l.], 1995. v. 4, p. 1942–1948.

KENNEDY, J.; MENDES, R. Population structure and particle swarm performance. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. [S.l.: s.n.], 2002. v. 2, p. 1671–1676 vol.2.

KIRK, D. B.; WEN-MEI, W. H. *Programming massively parallel processors: a hands-on approach*. [S.l.]: Morgan kaufmann, 2016.

KOH, B.-I. et al. Parallel asynchronous particle swarm optimization. *International journal for numerical methods in engineering*, Wiley Online Library, v. 67, n. 4, p. 578–595, 2006.

KUI-TING; CHEN. Hardware architecture of high-speed particle swarm optimization algorithm with programmable capabilities. 2014.

KUMAR, J.; SINGH, L.; PAUL, S. Gpu based parallel cooperative particle swarm optimization using c-cuda: A case study. In: . [S.l.: s.n.], 2013. v. 2013, p. 1–8. ISBN 978-1-4799-0020-6.

LALWANI, S. et al. A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering*, v. 44, 01 2019.

LI, S.-A. et al. Hardware/software co-design for particle swarm optimization algorithm. In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2010. p. 3762–3767.

LU, Y. et al. Study of distance vector routing protocols for mobile ad hoc networks. In: IEEE. *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003)*. [S.l.], 2003. p. 187–194.

MOLGA, M.; SMUTNICKI, C. Test functions for optimization needs. *Test functions for optimization needs*, v. 101, p. 48, 2005.

PASRICHA, S.; DUTT, N. *On-chip communication architectures: system on chip interconnect*. [S.l.]: Morgan Kaufmann, 2010.

RICHARDS, M.; VENTURA, D. Choosing a starting configuration for particle swarm optimization. *Neural Networks*, p. 2309–2312, 2004.

ROSENBROCK, H. H. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, v. 3, n. 3, p. 175–184, 01 1960. ISSN 0010-4620. Disponível em: <https://doi.org/10.1093/comjnl/3.3.175>.

RUARO, M. et al. Memphis: a framework for heterogeneous many-core socs generation and validation. *Design Automation for Embedded Systems*, v. 23, 12 2019.

SCHUTTE, J. F. et al. Parallel global optimization with the particle swarm algorithm. *International journal for numerical methods in engineering*, Wiley Online Library, v. 61, n. 13, p. 2296–2315, 2004.

SHI, Y.; EBERHART, R. C. Empirical study of particle swarm optimization. In: IEEE. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. [S.l.], 1999. v. 3, p. 1945–1950.

SHI, Y. et al. Particle swarm optimization: developments, applications and resources. In: IEEE. *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*. [S.l.], 2001. v. 1, p. 81–86.

SULZBACH, M. et al. Programação paralela híbrida para cpu e gpu: Uma avaliação do openacc frente a openmp e cuda. Universidade Federal de Santa Maria, 2014.

SURVANOVICK, D. B. S. Virtual libraryos simulation experiments:test functions and datasets. 2013. Disponível em: <https://www.sfu.ca/ ssurjano/rosen.html>.

TEWOLDE, G. S.; HANNA, D. M.; HASKELL, R. E. A modular and efficient hardware architecture for particle swarm optimization algorithm. *Microprocessors and Microsystems*, v. 36, n. 4, p. 289–302, 2012. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S014193311200018X>.

TOLEDO, L. et al. Towards enhancing coding productivity for gpu programming using static graphs. *Electronics*, v. 11, n. 9, 2022. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/11/9/1307>.

VENTER, G.; SOBIESZCZANSKI-SOBIESKI, J. Particle swarm optimization. *AIAA Journal*, v. 41, p. 1583–1589, 2002. Disponível em: <https://api.semanticscholar.org/CorpusID:4684741>.

VENTER, G.; SOBIESZCZANSKI-SOBIESKI, J. Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, v. 3, n. 3, p. 123–137, 2006.

WAINTRAUB, M.; SCHIRRU, R.; PEREIRA, C. Parallel particle swarm optimization algorithms in nuclear problems. In: (ABEN), A. B. de E. N. (Ed.). *Proceedings of the International Nuclear Atlantic Conference (INAC 2009)*. [S.l.], 2009. v. 1, p. 1–12.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor system-on-chip (mpsoc) technology. *IEEE transactions on computer-aided design of integrated circuits and systems*, IEEE, v. 27, n. 10, p. 1701–1713, 2008.

WOLPERT, D. H.; MACREADY, W. G. Coevolutionary free lunches. *IEEE Transactions on evolutionary computation*, IEEE, v. 9, n. 6, p. 721–735, 2005.

XU, F. et al. A custom parallel hardware architecture of nonlinear model-predictive control on fpga. *IEEE Transactions on Industrial Electronics*, v. 69, n. 11, p. 11569–11579, 2022.