

UNIVERSIDADE FEDERAL DE UBERLÂNDIA - UFU
FACULDADE DE ENGENHARIA ELÉTRICA - FEELT
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Geraldo Dutra Neto

**Paralelismo De Um Algoritmo
Genético, Aplicado a Otimização de
Rotas em Mineração De Modo Escalável**

Uberlândia
2018

Geraldo Dutra Neto

**Paralelismo De Um Algoritmo
Genético, Aplicado a Otimização de
Rotas em Mineração De Modo Escalável**

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Sistemas Embarcados

Orientador: Dr. Fábio Vincenzi Romualdo da Silva

Coorientador: Dr. Aniel Silva de Morais

Uberlândia

2018

Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/UFU que aloja o Programa de Pós-Graduação de Engenharia Elétrica.

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

D978p
2018 Dutra Neto, Geraldo, 1981-
 Paralelismo de um algoritmo genético, aplicado a otimização de rotas em mineração de modo escalável / Geraldo Dutra Neto. - 2018.
 76 f. : il.

 Orientador: Fábio Vincenzi Romualdo da Silva.
 Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
 Disponível em: <http://dx.doi.org/10.14393/ufu.di.2018.1138>
 Inclui bibliografia.

 1. Engenharia elétrica - Teses. 2. Algoritmos genéticos - Teses. 3. Otimização estrutural - Teses. 4. Mineração a céu aberto - Teses. I. Silva, Fábio Vincenzi Romualdo da, 1974- II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 621.3



SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



Ata da defesa de DISSERTAÇÃO DE MESTRADO junto ao Programa de Pós-graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia.

Defesa de Dissertação de Mestrado Acadêmico, número 681/2018/PPGEE

Data: 23 de abril de 2018

Discente: Geraldo Dutra Neto

Número de matrícula: 11522EEL005

Título do Trabalho: PARALELISMO DE UM ALGORITMO GENÉTICO, APLICADO NA OTIMIZAÇÃO DE POTAS EM MINERAÇÃO DE MODO ESCALÁVEL

Área de concentração: Sistemas de Energia Elétrica

Linha de pesquisa: Controle e Automação

As 08:00 horas do dia 23 de abril de 2018 na Sala de Defesas da Faculdade de Engenharia Elétrica, Campus Santa Mônica da Universidade Federal de Uberlândia, reuniu-se a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta:

Fábio Vincenzi Romualdo da Silva

CPF: 191.533.808-51, orientador

Kleber Lopes Fontoura

CPF: 542.973.541-87

Aniel Silva de Morais

CPF: 044.963.036-63, coorientador

Iniciando os trabalhos o presidente da mesa Prof. Dr. Fábio Vincenzi Romualdo da Silva apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu os conceitos finais.

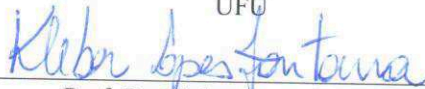
Em face do resultado obtido, a Banca Examinadora considerou o candidato A provado.

Esta defesa de Dissertação de Mestrado Acadêmico é parte dos requisitos necessários à obtenção do título de Mestre. O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar, foram encerrados os trabalhos às 10 horas e 50 minutos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.


Prof. Dr. Fábio Vincenzi Romualdo da Silva
UFU


Prof. Dr. Aniel Silva de Morais
UFU


Prof. Dr. Kleber Lopes Fontoura
CEFET

*Este trabalho é dedicado a minha família,
e especialmente a minha esposa Larissa
e ao meu filho Mateus
que são minhas motivações da vida.*

Agradecimentos

Os autores agradecem ao UFU - Universidade Federal de Uberlândia e a Instale Tecnologia pela disponibilidade de pesquisas, desenvolvimento e testes da aplicação.

*“A coisa mais indispensável a um homem
é reconhecer o uso que deve fazer
do seu próprio conhecimento.”*

(Platão)

Resumo

Neto, G.D. **Paralelismo De Um Algoritmo Genético, Aplicado a Otimização de Rotas em Mineração De Modo Escalável.** 77 p. Dissertação de mestrado – Faculdade de Engenharia de Uberlândia, Universidade de Uberlândia, 2018.

Atualmente, a tendência tecnológica está voltada para inteligência artificial e Internet das Coisas aplicadas em sistemas embarcados. Com a constante melhora de processos que estão sendo implantados como cultura empresarial na maior parte das empresas, a área de operação de mina no setor de mineração, não ficou de fora da busca por novas tecnologias. Entre os principais focos de melhorias, está o sistema de despacho, que atua na gestão de frota. O alto custo de um sistema de otimização de rotas de um sistema de despacho ainda não favorece a sua implantação em todas as mineradoras. Desse modo, este trabalho busca otimizar o tempo de execução de um algoritmo genético desenvolvido especificamente para a otimização de rotas em minas a céu aberto. O algoritmo genético proposto, utiliza paralelização com modelo tipo Ilha e usa a ferramenta OpenMP para realizar a programação multi-processo para ser aplicado em mineração de pequeno, médio e grande porte na otimização de despacho em mineração com múltiplas rotas. O gerenciamento das threads com o processo de execução do algoritmo genético necessita de dados específicos para a execução, onde nessa proposta tem configuração escalável de acordo com o porte da mineradora.

Palavras-chave: Sistemas Embarcados, Otimização de Rotas, Mineração, Sistema de Despacho, Sistemas Paralelos, Algoritmo Genético.

Abstract

Neto, G.D. **Parallel Scalability of an Applied Genetic Algorithm in Embedded Systems for Route Optimization.** 77 p. Master Thesis – Uberlândia School of Engineering, University of Uberlândia, 2018.

At present, the technological trend is focused on artificial intelligence and Internet of Things applied in embedded systems. With the constant improvement of processes that are being implemented as a business culture in most companies, the mine operating area in the mining sector has not been left out of the search for new technologies. Among the main focuses of improvements is the dispatch system, which operates in fleet management. The high cost of a route optimization system for a dispatch system still does not favor its deployment in all mining companies. This work seeks to optimize the execution time of a genetic algorithm developed specifically for the optimization of routes in open pit mines. The proposed genetic algorithm uses parallelization with the Island-type model and uses the OpenMP tool to perform the multi-process programming to be applied in small, medium and large-scale mining in the optimization of dispatch in mining with multiple routes. The management of the threads with the process of execution of the genetic algorithm requires specific data for execution, where in this proposal it has a scalable configuration according to the size of the mining company.

Keywords: Embedded systems, Optimization of Routes, Mining, Dispatch System, Parallel Systems, Genetic Algorithm.

Lista de ilustrações

Figura 1	Sistema Embarcado de Despacho em Equipamento de Transporte . . .	20
Figura 2	Sincronização entre Processos	24
Figura 3	Monothread	26
Figura 4	Processo Multithread	26
Figura 5	Granularidades	32
Figura 6	Modelo OpenMP	35
Figura 7	Fluxograma do Algoritmo Genético Sequencial	42
Figura 8	Algoritmo Panmitic	43
Figura 9	Fluxograma do Algoritmo Panmitic	45
Figura 10	Fluxograma do Algoritmo Granularidade Fina	46
Figura 11	Fluxograma do Algoritmo Island	48
Figura 12	Mineração a Céu Aberto	51
Figura 13	Carregamento de Minério - Frente de Lavra	52
Figura 14	Ciclo de Transporte com Fila no Carregamento	53
Figura 15	Cerca Eletrônica na Tomada de Decisão	54
Figura 16	Diagrama de Mapeamento de Rotas de Ponto Origem e Destino	55
Figura 17	Mapa Real da Mina de Fostato	56
Figura 18	Exemplo de Melhor Rota	57
Figura 19	Imagem da interface da solução proposta por Torres (2017)	58
Figura 20	Sistema embarcado no equipamento recebendo mensagem de alteração de rota.	59
Figura 21	Raspberry PI 2	60
Figura 22	Raspberry PI 3	60
Figura 23	Banana PI	61
Figura 24	Mina de Pequeno Porte - Banana Pi	66

Figura 25	Mina de Médio Porte - Banana Pi	66
Figura 26	Mina de Grande Porte - Banana Pi	67
Figura 27	Mina de Pequeno Porte - Raspberry Pi2	68
Figura 28	Mina de Médio Porte - Raspberry Pi2	68
Figura 29	Mina de Grande Porte - RaspberryPi2	69
Figura 30	Mina de Pequeno Porte - Raspberry Pi3	70
Figura 31	Mina de Médio Porte - Raspberry Pi3	70
Figura 32	Mina de Grande Porte - Raspberry Pi3	71
Figura 33	Melhor resultado de desempenho	71
Figura 34	Gráfico de resultado geral por tempo - mina de pequeno porte	72
Figura 35	Gráfico de resultado geral por tempo - mina de médio porte	72
Figura 36	Gráfico de resultado geral por tempo - mina de grande porte	73

Lista de tabelas

Tabela 1	Especificação Banana Pi M3	65
Tabela 2	Especificação Raspberry Pi2	67
Tabela 3	Especificação Raspberry Pi3	69

Lista de siglas

ARB OpenMP Architecture Review Board

API Application Programming Interface

ARM Advanced RISC Machine

AG Algoritmo Genético

AGP Algoritmos Genéticos Paralelos

CUDA Compute Unified Device Architecture

CPU Central Processing Unit

CSV Comma-separated values

FMT Fine-grained Multithreads

GPU Graphics Processing Unit

ILP - Instruction Level Parallelism

IOT - Internet of Things

IEEE Institute of Electrical and Electronics Engineers

MPI Message Passing Interface

OpenMP Open Multi-Processing

PVM Parallel Virtual Machine

Pthread POSIX threads

PLC Programmable logic controller

POSIX Portable Operating System Interface

PO Pesquisa Operacional

RNA redes neurais artificiais

SPMD Single Program Multiple Data

SMP Symmetric Multi-Processing

SIMD Single Instruction, Multiple Data

TLP Thread-Level Parallelism

TMU Threads em Modo de Usuário

VRP Vehicle Routing Problem

Sumário

1	Introdução	15
1.1	Motivação e Objetivo	16
1.2	Estado da Arte	16
2	Sistemas Embarcados	19
3	Computação Paralela	22
3.1	Unidades de paralelismo: Threads e processos	23
3.1.1	Processos	23
3.1.2	Threads	24
3.2	Ferramentas de Paralelismo	28
3.2.1	Softwares	32
3.2.2	Hardware	37
4	Algoritmos Genéticos	41
4.0.1	Algoritmos Genéticos Paralelos	42
5	Busca de Rotas	50
5.1	Problema na Mineração	50
6	Metodologia	58
6.0.1	Escalabilidade	61
7	Resultados	64
8	Discussão e Conclusões	74
8.1	Sugestões de Trabalhos Futuros	74
	Referências	76

Introdução

O homem está sempre tentando encontrar a melhor forma de realizar algum trabalho, e utiliza-se da otimização como uma ferramenta importante na execução de qualquer atividade na busca para alcançar objetivos da forma mais econômica possível. Com isso o campo de otimização vem sendo muito estudado gerando uma grande quantidade de estudos nessa área. (ALVARENGA, 1997).

A inteligência artificial está entre as tendências para tecnologia de 2018, e dentre elas a forma de trabalhar com os dados e será incorporada à - Internet of Things (IOT) (Internet das Coisas) e com o custo de processamento e de energia do consumo dos equipamentos diminuindo, em breve, haverá 100 bilhões de dispositivos conectados e, rapidamente, 1 trilhão. A magnitude da combinação de dados, poder de processamento com o poder da Inteligência Artificial, vai ajudar as máquinas a orquestrarem melhor os recursos físicos e humanos.(COMPUTERWORLD, Acessado em 15/12/2017).

Por sua vez, a inteligência computacional em sistemas embarcados ainda é muito pouca explorada, principalmente na área de mineração. A evolução dos hardwares vem acontecendo em uma velocidade exponencial, onde cada vez mais a sua capacidade de processamento podem ser utilizadas para o desenvolvimento de softwares com capacidade de resolver problemas do cotidiano ou até mesmo de alguma área importante da indústria, logística ou outras.

A utilização de Algoritmo genético, que é uma solução de meta-heurística pode ser uma alternativa de encontrar uma solução ótima local ou global dentro de um espaço de busca que se baseia metaforicamente na natureza, onde indivíduos mais bem adaptados permanecem para as próximas gerações e seus descendentes tendem a melhorar sua aptidão ainda mais através de cruzamentos e mutações. Podemos optar com a paralelização do algoritmo para a melhoria de desempenho, podendo assim otimizar os recursos existentes de arquitetura ou expandir os casos de utilização do algoritmo. Os Algoritmos Genéticos Paralelos (AGP) podem utilizar a arquitetura paralela dos processadores comercializados atualmente com diversos núcleos, e a natureza paralela da aplicação para obter ganho de desempenho sobre a versão sequencial. Linden (2012)

A necessidade até o momento, de solucionar esses problemas, dependiam de hardwares caros, como servidores de aplicação, nos quais as aplicações multitarefas utilizam de processamento paralelo. (YAGHMOUR et al.,).

A utilização de algoritmos genéticos em busca de melhor rota na mineração ainda é pouco explorada. Porém quando utilizada é com o processamento da solução ótima dentro de um robusto servidor com custo alto de aquisição e manutenção. Essa solução é enviada para os caminhões através de mensagem, por uma rede de alta capacidade, e devem ser recebido e interpretado no momento correto de sugestão de rota para caminhão, que fisicamente está entre um cruzamento de duas ou mais rotas, enviando assim o equipamento para um local onde a produtividade pode ser aumentada, evitando filas de carregamento ou descarregamento de material. (TORRES, 2017)

1.1 Motivação e Objetivo

A Lei de Gordon Moore, Moore (1965), previa um crescimento exponencial de transistores dos chips - 100 a cada 18 meses - pelo mesmo custo. Este crescimento traduzia em ganhos exponenciais de velocidade do Clock e no número de instruções executadas. Porém este crescimento não pode mais, até o momento, superar os limites físicos e problemas como geração e dissipação de calor além do alto consumo de energia. Existem ainda problemas relativo ao custo no desenvolvimento de pesquisas para reduzir componentes, sem quebrar a compatibilidade do modelo atual.

Técnicas que otimizem o tempo de processamento, algoritmos mais eficientes e computadores mais rápidos, abrem novos horizontes possibilitando realizar tarefas que antes eram inviáveis ou mesmo levariam muito tempo para serem concluídas, Sutter e Larus (2005), em seu artigo “The free lunch time is over”, afirma que não podemos mais esperar que nossos algoritmos fiquem mais rápidos apenas com atualizações de CPU. Muitos aplicativos se aproveitaram do fenômeno do “almoço grátis”, com ganhos regulares de desempenho por décadas, simplesmente esperando novas versões de CPU, memórias e discos mais rápidos.

A otimização de rotas a partir de um ponto de origem até o destino na mineração é um problema bem conhecido e possui várias soluções como, o algoritmo Dijkstra, Bellman-Ford, etc. A maioria das soluções também são apresentadas através de soluções de programação linear e programação dinâmica trabalhando juntas em servidores de grande porte com custos elevados e de grande desempenho(ALVARENGA, 1997).

1.2 Estado da Arte

Nesse sentido, este trabalho tem como objetivo paralelizar um algoritmo genético que foi desenvolvido em conjunto com Maicon Fernandes Torres para ser usado em uma

rede de sistemas embarcados que utilizam algoritmos genéticos aplicado na otimização de despacho em mineração a céu aberto com múltiplas rotas. (TORRES; NETO, 2016)

No entanto, Torres (2017) apresentou uma proposta de otimização que dispensa o uso de servidores¹ para a execução do algoritmo de tomada de decisão. Para pequenas mineradoras, o investimento em servidores pode ser inviável. Nesse caso, a solução apresentada viabiliza uma solução dentro do mercado existente que pode chegar a custo de milhões de reais de economia para uma mineradora de pequeno porte.

Com o paralelismo do algoritmo proposto, pretende-se utilizá-lo em mineradoras de grande porte, de modo a contribuir para melhor utilização do hardware, de modo a se obter redução de custo de implementação e melhor eficiência, isto é, redução do tempo de execução do algoritmo.

Assim, este trabalho visa explorar a eficiência do paralelismo de algoritmo genético em sistemas embarcados. O Modelo de Ilhas (Island) é utilizado para realizar o paralelismo do algoritmo genético proposto. A solução proposta fará uso da ferramenta OpenMP (Open Multi-Processing) para implementar a programação multi-processo.

Os resultados experimentais foram obtidos por meio de dados reais de mineradoras de pequeno, médio e grande porte.

O Capítulo 2 trata de sistemas embarcados e a constante evolução de suas arquiteturas, conforme citado por YAGHMOUR et al. (). Nesse capítulo, são apontadas as características das arquiteturas mais adequadas para o uso em mineração, considerando-se um ambiente severo, conforme Torres (2017).

O Capítulo 3 fala sobre computação paralela conforme Quinn (2003) e Almasi e Gottlieb (1989). Nesse capítulo, o conceito de paralelismo computacional para distribuição de tarefas entre processadores é descrito segundo Linden (2012), que aborda conceitos básicos das unidades de paralelismo. Continuando, Rosário (2012) explica o que são threads e processos com detalhes sobre de sincronismo e desenvolvimento em ambiente multithread, que são essenciais para o entendimento de paralelismo deste trabalho. Anderson et al. (1991) explica o desenvolvimento sobre paralelismo em modo de usuário e modo de kernel. Quinn (2003) apresenta os paradigmas de paralelismo computacional e demonstra com alguns exemplos as ferramentas de paralelismo, desde os recursos de software com ferramentas conhecidas como Open Multi-Processing (OpenMP) e Message Passing Interface (MPI) até o paralelismo de hardware usando Compute Unified Device Architecture (CUDA) e NEON.

O Capítulo 4 aborda o conceito de algoritmos genéticos segundo Linden (2012) e explica sobre os modelos de paralelismo de algoritmo genético mais conhecidos. O Modelo Ilha, apresentado por Ochi et al. (1998) é utilizado para realizar o paralelismo do algoritmo genético proposto por (TORRES; NETO, 2016). Esse capítulo também trata sobre o

¹Computador robusto, caro e que apresentam elevado poder de processamento, fabricado para operar de modo ininterrupto.

trabalho de Whitley, Rana e Heckendorn (1998) que executou o paralelismo de um algoritmo genético com o modelo de Ilha com e sem operador de migração, que no caso desse projeto será aplicado com o modelo original de Ochi et al. (1998).

O Capítulo 5 explica o problema de mineração que o algoritmo genético proposto é utilizado para resolver e comenta a visão de Alvarenga (1997) sobre as exigências de obtenção de um resultado eficiente para mineração e também, onde Torres (2017) utiliza o algoritmo embarcado em uma mineradora de pequeno porte e que nesta proposta detalha mais ainda o problema de buscas de melhor de rotas na mineração.

O Capítulo 6 apresenta a metodologia do trabalho proposto, onde é reforçado o problema da mineração na busca de melhores rotas. Comenta também como o algoritmo proposto é executado no sentido de se obter bons resultados.

O Capítulo 7 apresenta os resultados obtidos com o algoritmo genético proposto paralelizado e executado em diferentes hardwares.

O Capítulo 8 apresenta as conclusões e discussões do trabalho proposto e sugestões para trabalhos futuros.

Sistemas Embarcados

Atualmente existe uma forte tendência de se utilizar sistemas embarcados em equipamentos de automação residencial, industrial, eletrodomésticos, carros, aeronaves, equipamentos militares, de telecomunicação, dentre outros. Os automóveis atuais, por exemplo, possuem vários módulos embarcados: injeção eletrônica, suspensão eletrônica, controle de emissão de poluentes, freios ABS, piloto automático, computador de bordo, entre outros sistemas. Além disso, os diversos módulos automotivos estão interligados podendo receber ou enviar informações entre si, de modo que seus esforços sejam coordenados e adequados em relação ao estado do automóvel. (YAGHMOUR et al.,)

No campo da mineração os sistemas embarcados são utilizados em vários equipamentos como: moinhos, fornos, caldeiras, dutos, esteiras transportadoras, caminhões, escavadeiras, sistema de despacho, dentre outros. A robustez do hardware é um requisito importante, pois na mineração as condições de temperatura, umidade, poeira e vento variam muito e podem desgastar ou danificar o equipamento e, desse modo, comprometer a operação e a segurança das instalações eletro-eletrônicas. Nas minas, os sistemas embarcados devem ser devidamente protegidos de poeira, umidade, choques mecânicos e variações de temperatura. Por isso, esses equipamentos não utilizam sistemas de refrigeração com ventilação forçada (cooler).

A Figura 1 mostra um sistema embarcado de despacho instalado em um caminhão de 90 toneladas. Ele consiste de um processador Intel Atom x86, 4 GB de memória DDR3 e HD de 40 GB SSD, tela touch screen, não possui ventilação forçada, não possui orifícios que permitem a entrada de poeira e pode ser alimentado com tensão contínua de 10 a 32V. Esse equipamento é utilizado para coletar temperatura do motor, rpm, velocidade do veículo, consumo de combustível, situação do tanque, situação do óleo do motor, dados de posicionamento por GPS e transmite as informações por rede WiFi ou GPRS. É um equipamento robusto, a prova de poeira e de umidade.

Na aplicação proposta, pretende-se utilizar um sistema embarcado para realizar o paralelismo de um algoritmo genético, aplicado a otimização de rotas em mineração de modo escalável. Nesse caso, o sistema embarcado deve possuir microprocessador com mais



Figura 1 – Sistema Embarcado de Despacho em Equipamento de Transporte

Fonte: Acervo do Autor

de um núcleo (TORRES, 2017), de modo a permitir a paralelização do algoritmo genético. Considerando ainda que as oscilações de tensão no sistema de alimentação dos caminhões de 35 toneladas, utilizados na mineração, podem variar de 8 a 19V, o sistema embarcado deve operar nessa faixa de tensão.

Uma mina possui várias opções de origens de extração de minério e de destinos para onde se leva o minério para processamento. Dentre as várias opções de origens e de destinos existem ainda outras tantas opções de cruzamentos entre os vários caminhos possíveis.

Ademais, a quantidade de caminhões utilizados no transporte do minério também varia. Nesse sentido, a quantidade de cromossomos do algoritmo genético também irá variar, dependendo do número de possibilidades de rotas. Cada caminhão representa um indivíduo e cada indivíduo pode percorrer um determinado número de rotas (cromossomos).

Quando se deseja achar a melhor rota para um determinado caminhão (caminhão alvo), as rotas que os outros caminhões estão seguindo no momento devem ser considerados para efeito de cálculo. O algoritmo genético deve levar em consideração todas as rotas (cromossomos) de modo a fornecer como resultado final a melhor rota para o caminhão

alvo. Dessa maneira, o termo escalável utilizado nesse trabalho, refere-se a quantidade de caminhões que será utilizada em um determinado momento. Visto que essa quantidade terá impacto no número de rotas (cromossomos) que deverão ser processadas pelo algoritmo genético.

Computação Paralela

Computação paralela é uma forma de computação em que vários cálculos são realizados ao mesmo tempo, operando sob o princípio de que grandes problemas geralmente podem ser divididos em problemas menores para que possam ser resolvidos concorrentemente (em paralelo). Essa técnica é utilizada em processamento de imagens, para resolver problemas de evolução galáctica, modelagem climática, dentre outras áreas. (QUINN, 2003)

A ideia do paralelismo computacional é proveniente da execução de tarefas na vida cotidiana. Tomando a construção de uma casa como exemplo: percebe-se que algumas tarefas podem ser realizadas em paralelo tais como o encanamento e a fiação elétrica, mas as paredes têm de ser erguidas antes da realização da fiação elétrica. Do mesmo modo, muitas vezes, na programação paralela a ordem de execução de tarefas restringe o paralelismo. (ALMASI; GOTTLIEB, 1989)

Cada processador tem um limite computacional que não pode ser rompido. A solução para aumentar esta capacidade seria comprar máquinas mais potentes, com maior poder de processamento, porém, nem sempre isso é possível, visto que o poder de processamento é finito.

Nesse caso, quando não é possível elevar o poder computacional, pode-se usar múltiplos processadores e/ou múltiplos núcleos. Nesse caso existem ferramentas de computação paralela, que podem ser utilizadas de modo a se aproveitar melhor os recursos computacionais, dividindo uma tarefa maior em várias sub-tarefas e executando-as de forma simultânea em vários núcleos. Com a evolução constante da tecnologia, o mercado está oferecendo cada vez mais, com custos cada vez mais acessíveis, recursos de hardware compatíveis com a computação paralela. (LINDEN, 2012)

Entre os anos de 1990 e 2000, houve a disseminação de microprocessadores com mais de um núcleo, que favoreceu o paralelismo de tarefas. Esta mudança de arquitetura foi consequência da estagnação da evolução de desempenho de microprocessadores singlecore. (ROSÁRIO, 2012)

O trabalho proposto tem como objetivo paralelizar o algoritmo genético de busca de melhor rota em um menor espaço de tempo. Nesse sentido, as threads e os processos

desempenham um papel importante para se alcançar esse objetivo.

3.1 Unidades de paralelismo: Threads e processos

Segundo Rosário (2012) Há duas abstrações comuns para descrever paralelismo: threads e processos.

3.1.1 Processos

O conceito de processo é mais abrangente do que ser entendido inicialmente como um programa em execução onde a implementação de um recurso compartilhado de processador, memória principal e dispositivos de entrada/saída são utilizados em um sistema multitarefa. O sistema deve controlar a execução de diversos programas e o uso concorrente do processador e demais recursos, sendo possível apenas porque o programa está associado a um processo. (MACHADO; MAIA, 1996)

Para que a concorrência ocorra sem problemas entre os programas, é necessário que todas as informações do programa interrompido sejam guardadas¹ para que quando ele volte a operar não lhe falte informação necessária a continuação do processamento. Para a aplicação de desenvolvimentos paralelos, a concorrência de processos não existe apenas pelo uso de processador pelos seus núcleos (core), como também a execução de processos em diferentes processadores.

Sincronização e Comunicação entre Processos Muitas vezes, em uma aplicação concorrente, é necessário que os processos se comuniquem entre si. Machado e Maia (1996). Diversos mecanismos, como variáveis compartilhadas na memória ou troca de mensagens, podem facilitar essa comunicação. Para isso é preciso que os processos concorrentes tenham uma execução sincronizada através de mecanismos do sistema operacional.

A Figura 2 demonstra a sincronização de processos onde um processo que está em estado de "gravação", ou seja, gravando o dado em um buffer, e passa para o estado de leitura. Entre esses existem um estado de sincronização correspondente a esse dado, para evitar que o processo trave e execute corretamente os seus estados entre gravação e leitura.

Os mecanismos que garantem a comunicação entre processos concorrentes e o acesso aos recursos compartilhados são chamados de "mecanismos de sincronização".

Apesar do termo processo ser utilizado na exemplificação de aplicações concorrentes, os termos subprocesso e thread têm o mesmo significado nesta abordagem.

¹É necessário que ocorra a troca de contexto que é o processo computacional de armazenar e restaurar o estado (contexto) de uma CPU de forma que múltiplos processos possam compartilhar uma única instância de CPU.

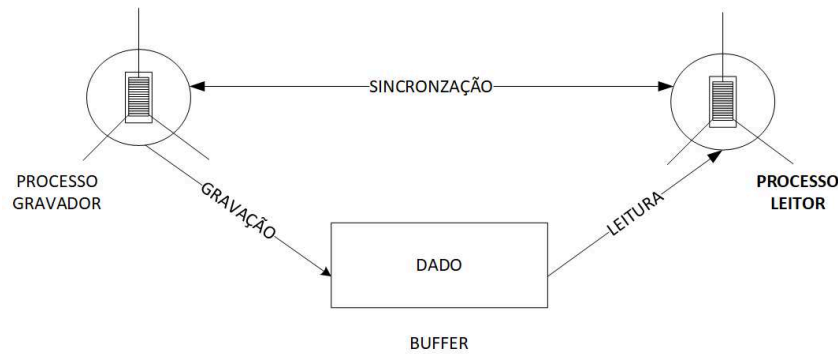


Figura 2 – Sincronização entre Processos

Fonte: Machado e Maia (1996)

3.1.2 Threads

A partir do conceito de múltiplos threads - multithread, é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter diferentes partes do seu código sendo executadas concorrentemente, com um menor overhead ² do que utilizando múltiplos processos. (MACHADO; MAIA, 1996)

O conceito de thread foi introduzido na tentativa de reduzir o tempo gasto em criação, eliminação e troca de contexto de processos nas aplicações concorrentes, bem como economizar recursos do sistema como um todo. Em um ambiente multithread, um único processo pode suportar múltiplos threads, cada qual associado a uma parte do código da aplicação. Como thread e processos ocupam o mesmo espaço de endereçamento, a comunicação entre threads não envolve mecanismos lentos de intercomunicação entre processos, aumentando, conseqüentemente, o desempenho da aplicação. O desenvolvimento de programas que exploram os benefícios da programação multithread não é simples. A presença do paralelismo introduz um novo conjunto de problemas, como a comunicação e sincronização das threads. Existe diferentes modelos para implementação de threads em um software ou sistema operacional, onde desempenho, flexibilidade e custo devem ser avaliados.

De forma simplificada, um thread pode ser definido como uma sub-rotina de um programa que pode ser executada de forma assíncrona, ou seja, executada concorrentemente ao programa chamador. Machado e Maia (1996) O programador deve especificar os threads, associando-se às sub-rotinas assíncronas.

A grande vantagem de uso threads é a possibilidade de minimizar a alocação de recursos do sistema, além de diminuir o overhead na criação, troca e eliminação de processos. O desempenho dos softwares pode ser beneficiados também com o uso de thread, diminuindo o tempo de execução e tornando possíveis aplicar inteligência em sistemas e softwares.

²Overhead é algum processamento ou armazenamento em excesso de um recurso que seja requerido para ser utilizado ou gasto para executar uma determinada tarefa.

Uma thread pode ser implementada no nível de usuário ou no nível do kernel ³. A vantagem da thread no espaço do usuário é que eles podem ser implementados sem a necessidade de alterar o núcleo. Antes de surgir o padrão Portable Operating System Interface (POSIX) para a implementação de threads, cada sistema operacional implementava sua própria versão de threads, fazendo com que fosse quase impossível para os programadores desenvolverem suas aplicações portáteis que usassem threads. Para superar esse problema foi desenvolvido o padrão POSIX 1003.1.c pelo Institute of Electrical and Electronics Engineers (IEEE) . A especificação POSIX não estabelece se os threads devem ser implementados no núcleo ou no espaço do usuário, e que define uma Application Programming Interface (API) com funções básicas de criação e controle de threads. (TANENBAUM, 2010)

No trabalho proposto a utilização de threads é essencial ao paralelismo do algoritmo genético aplicado na busca de melhor rota, fazendo assim suporte a sua escalabilidade, sendo que a ferramenta utiliza-se recurso de paralelismo de threads, conduzindo o processador a executar o algoritmo em menor tempo. Com o recurso multicore dos processadores, etapas da execução do algoritmo genético, podem ser divididas programavelmente para executar em outro núcleo do processador.

Monothread e Multithread Um programa é uma sequência de instruções, composta por desvios, repetições e chamadas a procedimentos e funções. Em um ambiente monothread, um processo suporta apenas um programa no seu endereçamento. Nesse ambiente, aplicações concorrentes são implementadas apenas com o uso de múltiplos processos independentes ou subprocessos. A utilização de processos e subprocessos independentes permite dividir uma aplicação em partes que podem trabalhar de forma concorrente. Um exemplo da concorrência pode ser encontrado nas aplicações com interface gráfica. Com o uso de múltiplos processos, cada funcionalidade do software implicaria a criação de um novo processo para atendê-la, aumentando o desempenho da aplicação. (MACHADO; MAIA, 1996)

O problema neste tipo de implementação é que o uso de processos no desenvolvimento de aplicações concorrentes demanda consumo de diversos recursos do sistema. Sempre que um novo processo é criado, o sistema deve alocar recursos para cada processo, consumindo tempo de processador nesse trabalho. No caso do término do processo, o sistema dispensa tempo para desalocar recursos previamente alocados.

A Figura 3 e a Figura 4 demonstram respectivamente uma monothread e um multithread, onde podemos observar que no ambiente monothread o espaço de endereçamento é individual, e não existe troca de contexto entre as threads, diferentes no ambiente mul-

³Um núcleo de sistema conecta o software aplicativo ao hardware de um computador. Em computação, o núcleo ou kernel é o componente central do sistema operativo da maioria dos computadores; ele serve de ponte entre aplicativos e o processamento real de dados feito a nível de hardware.

tithread, onde o espaço de endereçamento coloca as threads em mesma situação, fazendo a troca de contexto mais fácil entre elas.

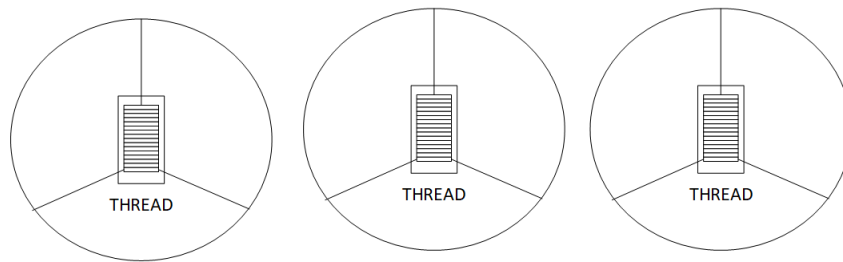


Figura 3 – Monthread

Fonte: Adaptado de Machado e Maia (1996)

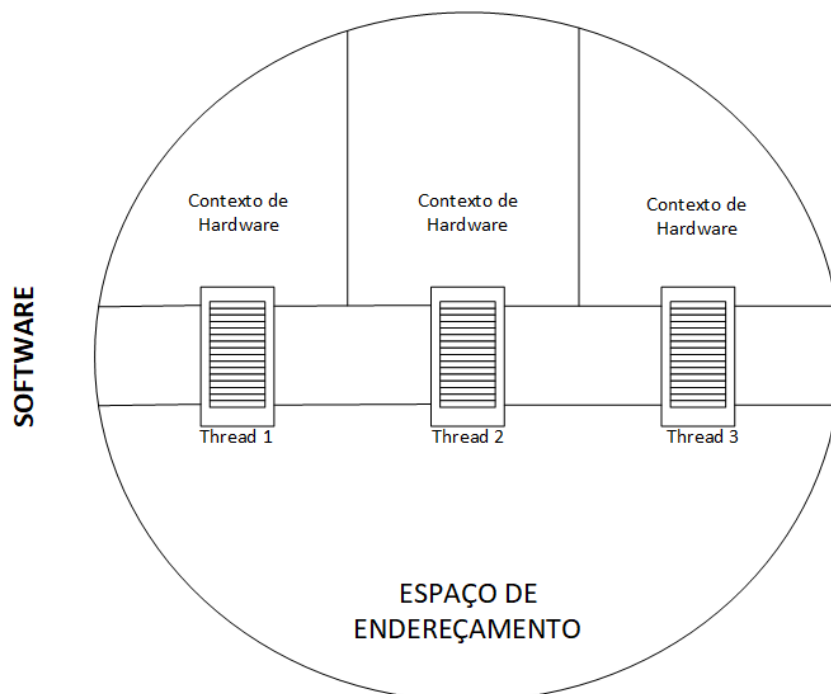


Figura 4 – Processo Multithread

Fonte: Adaptado de Machado e Maia (1996)

Desenvolvimento Multithread Existem diferentes abordagens na implementação dos pacotes de threads em uma aplicação, o que influenciará o desempenho, na concorrência e na modularidade das aplicações multithreads. Threads também ser oferecidos por bibliotecas de rotinas fora do núcleo do sistema operacional (modo usuário), pelo próprio núcleo do sistema (modo kernel) e uma combinação de ambos (modo híbrido). Também existe o modelo conhecido como 'scheduler activations'. (MACHADO; MAIA, 1996)

Uma das grandes dificuldades para a utilização de threads foi a ausência de um padrão. Em 1995, o padrão POSIX P1003.1c foi aprovado e posteriormente atualizado para a versão

POSIX 1003.4a. Com esse padrão, também conhecido por POSIX threads (Pthread), aplicações comerciais multithreading tornaram-se mais simples e de fácil implementação. O padrão Pthread é largamente utilizado em ambientes, UNIX, Linux e Solaris.

Thread Modo Usuário e Thread Modo Kernel De acordo com Machado e Maia (1996), Threads em Modo de Usuário (TMU) são implementadas pela aplicação e não pelo sistema operacional. Para isso deve-se existir uma biblioteca de rotinas, que possibilite à aplicação realizar tarefas como criação/eliminação e threads, troca de mensagens entre threads e uma política de escalonamento.

Com threads de modo usuário, o sistema operacional não sabe da existência de múltiplos threads, sendo responsabilidade exclusiva da aplicação gerenciar e sincronizar os diversos threads existentes.

A vantagem desse modelo é a possibilidade de implementar aplicações multithreads mesmo em sistemas operacionais que não suportam threads. A limitação desse sistema é que o sistema operacional trata cada processo como se existisse apenas uma thread. No momento que um thread chama uma rotina do sistema que o coloca em estado de espera, todo o processo é colocado no estado de espera, mesmo havendo outros threads prontos para execução. Para contornar essa limitação, a biblioteca usada para o desenvolvimento, tem que possuir rotinas que substituam as rotinas bloqueantes por outras que não possam causar o bloqueio de uma thread. Todo esse controle é transparente para o usuário e o sistema operacional. Em relação ao escalonamento em ambientes com múltiplos processadores, não é possível que múltiplos threads de um processo possam ser executados em diferentes CPU's simultaneamente, pois o sistema seleciona apenas processos para execução e não threads. Essa restrição limita dramaticamente o grau de paralelismo da aplicação, já que os threads de um mesmo processo podem ser executados em somente um processador de cada vez. (TANENBAUM, 2010)

Threads em modo kernel são implementados diretamente pelo núcleo do sistema operacional, através de chamadas de rotinas do sistema que oferecem todas as funções de gerenciamento e sincronização. O sistema operacional sabe da existência de cada thread e pode escaloná-las individualmente. No caso de múltiplos processadores, as threads de um mesmo processo podem ser executadas simultaneamente.

O grande problema para pacotes em modo kernel é o seu baixo desempenho. Enquanto nos pacotes no modo usuário todo tratamento é feito sem a ajuda do sistema operacional, ou seja, sem a mudança do modo de acesso, pacotes em modo kernel utilizam rotinas do sistema e, conseqüentemente, várias mudanças no modo de acesso. (ANDERSON et al., 1991)

No projeto proposto é utilizado a biblioteca OpenMP, de modo usuário, que gerencia as threads a serem executadas, sendo escolhida pela sua prática de ser aplicada e da melhor forma de gerenciamento das threads do algoritmo genético que será demonstrada

a forma de paralelização no capítulo de metodologia.

Thread Modo Híbrido A arquitetura de threads em modo híbrido combina vantagens de implementações em modo usuário e modo kernel. Um processo pode ter vários threads em modo kernel e, por sua vez, várias threads em modo de usuário dentro das threads de kernel. O núcleo do sistema reconhece as threads em modo de kernel e pode escaloná-los individualmente. Uma thread em modo usuário pode ser executada dentro de uma thread de modo de kernel, em um determinado momento, e no instante seguinte ser executado em outro. (MACHADO; MAIA, 1996)

Quando um desenvolvedor desenvolve a aplicação em termos de thread em modo de usuário e especifica quantos threads em modo de kernel estão associados ao processo e sendo mapeados.

O modo híbrido, de maior flexibilidade, apresenta problemas herdados de ambas as implementações. Por exemplo, quando uma thread em modo kernel realiza uma chamada bloqueante, todas as threads em modo usuário são colocados no estado de espera. Threads em modo de usuário que desejam utilizar vários processadores devem utilizar diferentes threads em modo de kernel o que influenciará no desempenho.

3.2 Ferramentas de Paralelismo

Diversos problemas matemáticos das mais diversas áreas estão sendo repensados e remodelados para novos paradigmas de programação paralela, tais como OpenMP, MPI, e CUDA, dentre outros. (ROSÁRIO, 2012)

A utilização das ferramentas de paralelismo tende a dificultar o desenvolvimento de uma aplicação, mas também possibilita uma redução relevante do tempo de execução imediato ou de acordo com o crescimento dos dados a serem processados.

A necessidade do processamento paralelo é explicada por vários fatores onde o principal deles trata da tentativa do maior desempenho. Em virtude das complexidades dos algoritmos utilizados, sejam eles científicos, industriais ou militares, um poder computacional é mais requisitado, considerando também um grande conjunto de dados a ser processado.

Um programa desenvolvido serialmente sempre será executado por apenas um core, por mais que a sua estrutura de execução esteja sendo em um processador multicore⁴ ou até mesmo sendo um cluster⁵.

A computação paralela pode ser feita em diferentes formas: bit, instrução, de dado ou de tarefa. Em Sistemas Embarcados, onde também é grande a preocupação com

⁴A palavra multicore é utilizada para definir qualquer processador que tenha mais de um núcleo.

⁵Um cluster consiste em computadores fracamente ou fortemente ligados que trabalham em conjunto que podem ser considerados como um único sistema

o consumo de energia, a computação paralela também se tornou um paradigma dominante. (ASANOVI et al., 2006)

O paralelismo de software ainda continua sendo o mais utilizados dos recursos de melhoria de desempenho de aplicações, sendo pela praticidade e evolução constante dos framework's⁶ e ferramentas de softwares.

Existem 4 maneiras básicas de se criar software paralelos:

1. Adicionar bibliotecas em linguagens sequenciais normais onde o paralelismo é restrito a alguns procedimentos e o núcleo do software permanece sequencial;
2. Adicionar bibliotecas com controle e gerenciamento de comunicação, que nesse caso o programador é responsável pela criação e gerenciamento do paralelismo dentro da linguagem de programação convencional;
3. Incorporar funções para execução paralela em linguagens de programação existente, que continua sendo a abordagem mais utilizada, e muitas linguagens possuem versões paralelas onde as funções permitem a geração de novos processos em paralelo;
4. Desenvolvimento de linguagens específicas para paralelismo, onde a grande vantagem é que a linguagem suporta todas as funções paralelas desejadas e a desvantagem é que os programadores precisam se familiarizar com uma nova linguagem.

Ao longo dos anos, uma infinidade de bibliotecas, extensões, funções e novas linguagens foram criadas e a classificação dessas tornou-se difícil. Temos 5 aspectos fundamentais que caracterizariam o software para máquinas paralelas:

Modelos de controle O aumento no tamanho de transistores, o crescimento de transferência de memória, as políticas de cachê gasto com energia faz que a exploração de paralelismo em nível de instrução - Instruction Level Parallelism (ILP) chegam em um limite no qual não é mais possível melhorar o desempenho processamento dentro das máquinas. Com a necessidade de se explorar o paralelismo, que se encontra em um nível acima do nível de instrução, foi desenvolvido o paralelismo em nível de thread, Thread-Level Parallelism (TLP).

Cada thread possui o seu próprio contador de programa, registradores e pilha, mas compartilha as variáveis globais com as outras threads. Cada thread é independente e roda em um processador diferente provavelmente.

A principal questão envolvida é se haverá uma ou várias threads de controle. No primeiro caso, uma instrução sendo executada em uma unidade de controle opera simultaneamente em vários dados. No segundo caso, várias instruções são executadas si-

⁶Um framework em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

multaneamente pelas respectivas unidades de controle, provavelmente comunicando-se e sincronizando-se umas com as outras que é o modelo predominante em paralelismo.

Paradigmas computacionais Programas paralelos necessitam de certos paradigmas para a sua estruturação. O primeiro paradigma refere-se à operação de uma thread de controle sobre um conjunto grande de dados. É chamado de Single Program Multiple Data (SPMD). Um segundo paradigma é o Pipeline, no qual os dados são operados por um 1º processo que os envia para um 2º processo e assim por diante. Outro paradigma estabelece uma computação em fases. Em cada fase, múltiplos processos operam em paralelo e é preciso esperar até que todos tenham terminado a operação para o início de uma nova fase. Tal paradigma é chamado de computação por fases. O quarto paradigma, denominado "dividir para conquistar", estabelece que um processo pode subdividir-se em outros processos para a divisão da carga de trabalho. Os processos assim gerados podem dividir-se subsequentemente. O último paradigma é conhecido como "replicated worker" ou também "task form", no qual uma fila de tarefas centralizada despacha as tarefas para os processos. Assim que um processo termina a sua computação ele busca uma nova tarefa na fila. Se durante a execução de um processo novas tarefas são geradas, estas são enviadas para a fila centralizada.

Métodos de comunicação Quando existem processos operando em paralelo, geralmente é necessária a comunicação entre os mesmos, o que pode ser feito de duas formas:

- Variáveis compartilhadas (Load, Store);
- Message Passing (Send, Receive).

Ainda com relação à troca de mensagens (message passing), é possível uma classificação quanto ao número de receptores de uma mensagem. Um primeiro caso é chamado point-to-point e a mensagem é dirigida a um único receptor. Quando a mensagem é dirigida a um conjunto específico de receptores, chama-se de multicasting. O último caso é o broadcasting, quando a mensagem é dirigida a todos os receptores.

A combinação de diferentes arquiteturas paralelas com estes métodos de comunicação é possível:

Multiprocessadores: comunicação por variáveis compartilhadas é padrão - a troca de mensagens pode ser simulada em memória. Multicomputadores: variáveis compartilhadas são possíveis em sistemas com uma camada de gerenciamento intermediária (DSM, Linda, Orca, etc) - troca de mensagens com Parallel Virtual Machine (PVM) ou MPI.

Primitivas de sincronização Além de comunicação, os processos muitas vezes necessitam de sincronização que pode ser exemplificada no acesso às estruturas de dados. Apenas um processo deve ter acesso aos dados enquanto estes são modificados. Uma das formas de garantir a integridade dos dados é chamada de exclusão mútua sendo a mais importante para evitar problemas de compartilhamento de recursos e impedir que

dois ou mais processos acessem ao um mesmo recurso simultaneamente. Para isso, enquanto um processo estiver acessando determinado recurso, todos os demais processos que queiram acessá-lo deverão esperar pelo término da utilização dos recursos. Essa ideia de exclusividade de acesso é chamada exclusão mútua (mutual exclusion).

A exclusão mútua deve afetar apenas os processos concorrentes somente quando um deles estiver fazendo acesso ao recurso compartilhado é denominado *região crítica*. Se for possível evitar exclusão mutuamente exclusiva das regiões críticas, os problemas decorrentes do compartilhamento serão evitados.

Os mecanismos que implementam a exclusão mútua utilizam processos de acesso à região crítica. Toda vez que um processo desejar executar instruções de sua região crítica, obrigatoriamente deverá executar um protocolo de entrada nessa região. Da mesma forma, ao sair da região crítica um protocolo de saída deve ser executado. Os protocolos de entrada e saída garantem a exclusão mútua da região crítica de um programa.

Várias primitivas de software são utilizadas para assegurar exclusão mútua:

- Semáforos - Locks - Mutexes - Critical sections

Uma outra primitiva de sincronização são as barreiras, que garantem o bloqueio de processos em uma fase até que todos os processos tenham realizado as suas respectivas computações.

Granularidade do paralelismo - Multithreads com granularidade fina, Fine-grained Multithreads (FMT), acontece quando a troca entre threads ocorre em curtos espaços de tempo. Uma vantagem nessa arquitetura, é que em casos de paradas, mesmo sendo curtas ou longas, fica oculta a perda em largura de banda, uma vez que, assim que uma thread está parada, as instruções de outra thread podem ser executadas tendo assim um balanceamento de carga mais eficiente. Essa arquitetura tem como desvantagem que: a execução de threads de forma individual pode ser atrasada, sendo que uma thread terá sua execução dependendo das outras threads.

Uma arquitetura é de granularidade grossa quando a troca para uma próxima thread ocorre pelo fato de que a thread que está em execução e utiliza uma operação de longa latência, como por exemplo, o acesso à memória principal devido a um uma perda de gerenciamento de cache. Neste tipo de arquitetura o desempenho de uma thread individual é menos afetado. Porém esta técnica traz como benefício os casos onde a parada é mais demorada que o tempo do preenchimento do pipeline. Neste tipo de arquitetura é mais difícil fazer um balanceamento eficiente de carga.

A Figura 5 demonstra o nível de detalhamento em um sistema paralelo através de granularidades, sendo que quanto mais fina a granularidade, mas existe interações entre threads em um sistema paralelo. Os detalhes de utilização dessas granularidades em um paralelismo de algoritmo genético serão utilizado o de modelo de granularidade grossa, chamado de modelo de ilha, descrito no capítulo 4 deste trabalho.

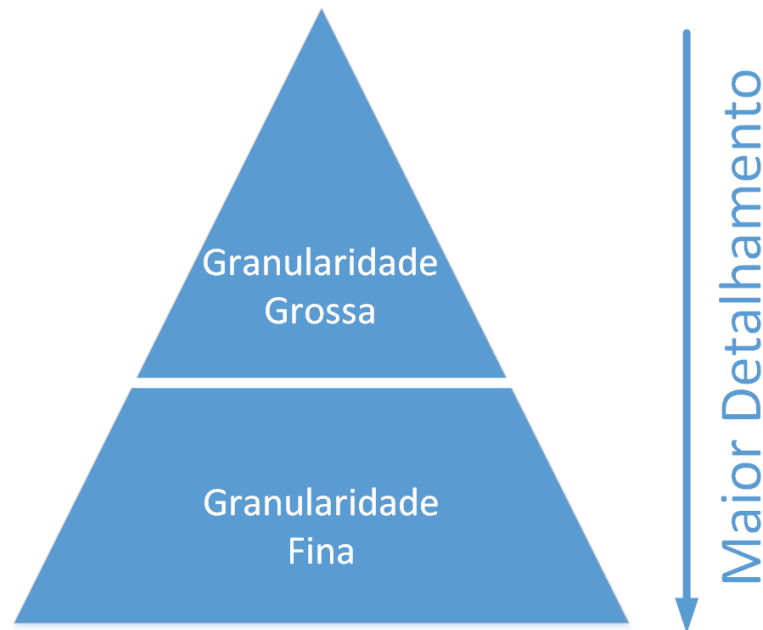


Figura 5 – Granularidades

Fonte: Acervo do Autor

3.2.1 Softwares

3.2.1.1 Pthread

Pthread são bibliotecas para as linguagens C e C++ que permitem gerar um conjunto de threads que poderão rodar em simultâneo. Implementações feitas com essas bibliotecas são mais eficientes em sistemas multiprocessador e/ou multicore onde os threads criados serão executados em múltiplos processadores/núcleos, possibilitando que o tempo de execução do programa diminua, ganhando mais velocidade através do processamento paralelo, este maior desempenho também pode ser obtido em sistemas com um único processador. A biblioteca Pthread trabalha com programação paralela com threading explícito, ou seja, o programador é responsável por criar as múltiplas threads e definir o trabalho que será realizado pelas mesmas. Além da criação das threads, o programador também deve gerenciar a sincronização entre as mesmas, para que a paralelização do código sequencial não corrompa os resultados obtidos.

Exemplo de Aplicação com Pthread.

```
1 ...  
2 #include "AGTools.h"  
3 #include <math.h>  
4 #include <pthread.h> // Inclusão da Biblioteca Pthread  
5 ...  
6  
7 const uint8_t
```

```
8 ...
9 double totalFitness
10 ...
11
12 #define MAX_THREADS 8 // Definição de Máximo de Threads Usadas
13 pthread_t Threads[MAX_THREADS]; // Lista de Threads
14 int ThreadRunning[MAX_THREADS];
15
16 //-----\\
17
18 void RandomPopulation()
19 {
20 uint16_t i=0;
21 generation = 0;
22 ClearVetor(population);
23
24 for(i=0;i<POP_SIZE;i++)
25 {
26 AddVetor(population,Random31());
27 }
28
29 for(i=0;i<MAX_THREADS;i++) // Inicialização das Threads
30 {
31 ThreadRunning[i] = 0;
32 }
33 }
34
35 double DecodeInd(uint64_t x)
```

O desenvolvimento de uma aplicação paralela com pthread está no conceito que adiciona bibliotecas especiais com primitivas de comunicação e controle, mesmo que tenha um controle sobre o paralelismo, o desenvolvimento tende a ser mais complexo. A grande vantagem é que o Pthread é uma biblioteca com portabilidade, podendo assim rodar em vários ambientes e sistemas operacionais.

Devido a maior complexidade de desenvolvimento do paralelismo, a biblioteca Pthread não foi utilizado neste trabalho, sendo que dentro as complexidades de desenvolvimento no gerenciamento das threads nos núcleos dos processadores do algoritmo genético escalável não compensaria pelo ganho proposto.

3.2.1.2 OpenMP

O OpenMP é uma interface de programação de aplicativos de memória compartilhada API cujos recursos são baseados em esforços anteriores para facilitar a programação paralela de memória compartilhada. Em vez de um padrão oficialmente sancionado, é um acordo alcançado entre os membros do OpenMP Architecture Review Board (ARB)⁷, que compartilham um interesse em uma abordagem portátil, amigável e eficiente para a programação paralela de memória compartilhada. O OpenMP destina-se a ser adequado para implementação em uma ampla gama de arquiteturas Symmetric Multi-Processing (SMP).

Como seus predecessores, o OpenMP não é uma nova linguagem de programação. Em vez disso, é uma notação que pode ser adicionada a um programa seqüencial em Fortran, C ou C++. A inserção apropriada dos recursos do OpenMP em um programa seqüencial permitirá que muitas, talvez a maioria, os aplicativos se beneficiem de arquiteturas paralelas - muitas vezes com modificação mínima do código.

O sucesso do OpenMP pode ser atribuído a vários fatores. Uma é a sua forte ênfase na programação paralela estruturada. Outro é que o OpenMP é comparativamente simples de usar, uma vez que o fardo de elaborar os detalhes do programa paralelo depende do compilador. Tem a maior vantagem de ser amplamente adotado, de modo que uma aplicação OpenMP seja executada em muitas plataformas diferentes.

```
1
2 ...
3 #include <omp.h> //Inclusão da Biblioteca OpenMP
4
5 int main()
6 ...
7
8 t_1 = omp_get_wtime(); //Retorno de Tempo
9
10 //Chamado do OpenMP
11 #pragma omp parallel for reduction(+: sum) schedule(static)
12 for (i = 0; i < N; ++i)
13 {
14 double x = x_0 + i * h + h/2;
15 sum += sqrt(1 - x*x);
16 }
17
18 t_2 = omp_get_wtime(); //Retorno de Tempo
19
```

⁷ARB - O OpenMP ARB (ou apenas ARB) É a empresa sem fins lucrativos que possui a marca OpenMP, supervisiona a especificação e produz e aprova novas versões da especificação

```

20 pi = sum * h * 4.0;
21
22 //Resultado de Execução
23 printf("omp_get_max_threads():_d\n", omp_get_max_threads());
24
25 ...
26
27 }

```

O OpenMP é baseado no modelo fork-join, conforme demonstrado na Figura 6 abaixo. Esse modelo executa a partir de uma estrutura serial, onde a partir da mudança de estrutura para fork distribui a execução em uma zona paralela para o ganho de desempenho. A partir disso, é necessário um processo de sincronização com uma estrutura de join.



Figura 6 – Modelo OpenMP

Fonte:Acervo do Autor

Pela facilidade de desenvolvimento e de portabilidade do OpenMP, foi a ferramenta escolhida para o desenvolvimento do paralelismo do algoritmo genético proposto, sendo que o gerenciamento de threads durante a execução do algoritmo pode ser facilmente escalonado entre os núcleos do processador tendo assim um ganho de tempo explicado melhor no capítulo de metodologia desse trabalho.

3.2.1.3 MPI

Ainda visando a necessidade de um padrão que implementa o modelo de passagem para computadores paralelos, clusters⁸ e multiprocessadores facilitando o acesso ao hardware paralelos aos usuários, foi projetado para auxiliar no desenvolvimento de softwares paralelos o padrão MPI. Segundo Pacheco (1996) a transmissão de mensagens tem a simples funcionalidade de transmitir dados de um processo para outro.

O modelo da API do MPI disponibiliza ao desenvolvedor uma camada de abstração, portátil, entre a tecnologia de comunicação ou sistema operacional e a aplicação, podendo assim permitir escalar a execução do programa entre os computadores na rede ou entre os núcleos do processador. Deve-ser ter em mente o desenvolvimento de um programa que realize a menor troca de mensagens possíveis, tendo como objetivo o ganho de desempenho.

⁸Um cluster (do inglês cluster : 'grupo, aglomerado') consiste em computadores fracamente ou fortemente ligados que trabalham em conjunto, de modo que, em muitos aspectos, podem ser considerados como um único sistema.

Conforme Quinn (2003), o MPI é o mais popular padrão de transmissão de mensagens de programação paralela. Praticamente todos os computadores paralelos comerciais suportam MPI, e as bibliotecas gratuitas que atendem ao padrão MPI estão disponíveis.

Exemplo de Aplicação com OpenMPI.

```
1
2 ...
3 #include <stdio.h>
4 #include <mpi.h> //Inclusão da OpenMPI
5 ...
6 char **argv;
7 {
8
9 int numero;
10 int myrank, size;
11
12 MPI_Status status; //Função MPI
13
14 MPI_Init(&argc,&argv);
15
16 MPI_Comm_rank(MPI_COMM_WORLD, &myrank); //Função MPI
17 MPI_Comm_size(MPI_COMM_WORLD, &size); //Função MPI
18
19 printf("Olá!...\n");
20
21 if (myrank == 0)
22 {
23 printf("Sou o processo 0\n");
24 scanf("%d", &numero);
25
26 MPI_Send(&numero,1,MPI_INT,1,99,MPI_COMM_WORLD); //Função MPI
27 MPI_Recv(&numero,1,MPI_INT,2,99,MPI_COMM_WORLD,&status);
28
29 ...
30 MPI_Finalize(); //Função MPI
31 }
```

MPI é uma interface especificada para sintaxe e semântica das operações de comunicação, mas deixa os detalhes da implementação abertos. Assim, diferentes bibliotecas MPI podem usar implementações diferentes, possivelmente usando otimizações específicas para

hardware em específicas plataformas. Para o programador, o MPI fornece uma interface padrão, garantindo assim a portabilidade dos programas MPI. (SNIR et al., 1998)

A utilização do MPI foi descartada neste projeto, sendo que essa ferramenta é melhor utilizada em sistemas de cluster, efetuando a troca de mensagens como se fosse uma única estrutura e não dentro de vários cores dentro de um mesmo processador, assim como funciona o OpenMP. Mesmo assim a facilidade de implementação do MPI ainda é melhor que o Pthread.

3.2.2 Hardware

Dentre as duas soluções apresentadas a seguir trazem os seus recursos de paralelismos baseados em arquiteturas de hardware e para a metodologia proposta neste trabalho não serão utilizadas devido que os mesmos não podem ser utilizados em uma solução com portabilidade sendo que estão dependentes dessa arquitetura, tando com CUDA que são dependentes de recursos apenas em produtos fabricados pela NVIDIA ou NEON que são apenas utilizados em alguns modelos de processadores Advanced RISC Machine (ARM). Mesmo com essas limitações, esse autor considera que os modelos de paralelismo devem ser conhecidos como opções de trabalhos futuros propostos.

3.2.2.1 CUDA

Ao longo da última década, a computação de alto desempenho evoluiu de forma significativa, particularmente devido ao surgimento de arquiteturas heterogêneas que levaram a uma mudança de paradigma fundamental na programação paralela. (CHENG; GROSSMAN; MCKERCHER, 2014)

CUDA, sigla para Compute Unified Device Architecture, é uma extensão para a linguagem de programação C, a qual possibilita o uso de computação paralela. A ideia por trás disso tudo é que programadores possam usar os poderes da unidade de processamento gráfico (GPU) para realizar algumas operações mais rapidamente. CUDA aproveita o poder da computação GPU para acelerar aplicativos.

O fluxo de processamento em CUDA não é tão complexo. Para começar, os dados são copiados da memória principal para a unidade de processamento gráfico. Depois disso, o processador aloca o processo para a Graphics Processing Unit (GPU), que então executa as tarefas simultaneamente em seus núcleos. Depois disso, o resultado faz o caminho inverso, ou seja, ele é copiado da memória da GPU para a memória principal.

Neste exemplo de aplicação com CUDA utilizamos também o MPI para poder melhor exemplificar uma multithread.

1
2
3

...

```
4 // Obtendo o número de nós.
5 int commSize, commRank;
6 MPI_CHECK(MPI_Comm_size(MPI_COMM_WORLD, &commSize));
7 MPI_CHECK(MPI_Comm_rank(MPI_COMM_WORLD, &commRank));
8
9
10 int dataSizeTotal = dataSizePerNode * commSize;
11 float *dataRoot = NULL;
12
13 ...
14
15
16 MPI_CHECK(MPI_Scatter(dataRoot,
17 dataSizePerNode,
18 MPI_FLOAT,
19 dataNode,
20 dataSizePerNode,
21 MPI_FLOAT,
22 0,
23 MPI_COMM_WORLD));
24
25 if (commRank == 0)
26 {
27
28 delete [] dataRoot;
29 }
30
31 //Coloca cada nó do MPI em cada GPU
32 computeGPU(dataNode, blockSize, gridSize);
33
34 // Reduz o nó raiz
35 float sumNode = sum(dataNode, dataSizePerNode);
36 float sumRoot;
37
38 MPI_CHECK(MPI_Reduce(&sumNode, &sumRoot, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD));
39
40 if (commRank == 0)
41 {
42 float average = sumRoot / dataSizeTotal;
```

```
43 cout << "Average of square roots is: " << average << endl;
44 }
45
46 // Cleanup
47 delete [] dataNode;
48 ..
```

A tecnologia CUDA apresenta dois pontos fracos em sua utilização que são limitações que devem ser levadas em conta pelos programadores ao desenvolverem seus softwares:

Não é suportada pelo CUDA, a renderização de texturas e as diversas cópias realizadas entre uma memória e podem causar algum impacto no desempenho geral das aplicações, variando de acordo com o barramento do sistema. Deve ser considerado a compatibilidade do software desenvolvido dessa forma.

CUDA é uma tecnologia da NVIDIA⁹ e, portanto, ela só existe com placas de vídeo fabricadas por essa empresa. Sendo pelo princípio de portabilidade a tecnologia CUDA não será utilizada nesse projeto.

3.2.2.2 Neon

O subsistema NEON é uma unidade de processamento de Single Instruction, Multiple Data (SIMD) avançada e significa que pode aplicar um único tipo de instrução a vários pedaços de dados ao mesmo tempo em paralelo. Isto é extremamente útil quando se trata de processamento de mídia, como filtros de áudio / vídeo e codecs. O sistema NEON não é a unidade de ponto flutuante do processador ARM.

Exemplo de Aplicação com NEON.

```
1 #include <stdio.h>
2 #include <arm_neon.h>
3
4 #define SIZE 64
5
6 int main(void){
7     uint32_t i;
8     uint8_t arr0[SIZE]; //Declaração de Variáveis NEON
9     uint8_t arr1[SIZE];
10    uint8_t arr2[SIZE];
11
12    for(i=0; i<SIZE; i++){
13        arr0[i] = i;
```

⁹NVIDIA é uma empresa multinacional de tecnologia com sede em Santa Clara, Califórnia que fabrica peças de computador, e é mais popularmente conhecida por sua série de placas de vídeo GeForce.


```
14 arr1[i] = i;
15 }
16
17 uint8x16_t arr0_;
18 uint8x16_t arr1_;
19
20 for(i=0; i<SIZE; i+=16){
21 arr0_ = vld1q_u8(arr0 + i); // arr0_ = arr0 //Utilização de Neon
22 arr1_ = vld1q_u8(arr1 + i); // arr1_ = arr1
23 arr1_ = vaddq_u8(arr1_, arr0_); // arr1_ = arr1_ + arr0_
24 vst1q_u8((arr2+i), arr1_); // arr2 = arr1_
25 }
26
27 ...
28
29 return 0;
30 }
```

A utilização da Tecnologia NEON depende do processador ARM ter a sua extensão e é limitada apenas a esta e ainda não suporta números de ponto flutuante de precisão dupla, o NEON só funciona em vetores e não suporta avançado operações como raiz quadrada e divisão. Devido a tecnologia NEON ser apenas utilizada em alguns tipos de processadores ARM ela não foi selecionada para esse trabalho sendo que não seria possível a portabilidade do software com os recursos paralelos e também considerando a complexidade do desenvolvimento.

Algoritmos Genéticos

A Teoria da evolução das espécies de Charles Darwin, trata que dentro de uma população, os indivíduos são possíveis soluções de um problema e assim foi inspirado a técnica dos Algoritmos Genéticos de Linden (2012).

Assim como na evolução, novos indivíduos podem ser gerados a partir de seleção e cruzamento de seus pais, preservando algumas características dos mesmos. A mutação de alguns genes também são primordiais para a evolução dos indivíduos. A partir de novas gerações uma população sempre substitui a antecessora. Os cruzamentos tendo como base uma função de avaliação, causam um aumento na diversidade genética e leva a melhor qualidade de soluções. Depois de um número de evoluções corretas, o algoritmo retorna o melhor indivíduo.

Para isso o algoritmo genético deve seguir as seguintes etapas, conforme demonstrado na Figura 7: (1) criar a população, onde cada indivíduo representa um conjunto contendo o ponto de origem, ponto de destino e a rota, (2) avaliação do fitness (avaliação de aptidão de cada indivíduo), (3) seleção dos indivíduos pais, (4) recombinação dos pais (gerando novos filhos), (5) mutação (evolução dos indivíduos). Após isso se repete a etapa na quantidade de vezes que for desejada tendo assim uma nova geração, ou seja, se for definido que o algoritmo deve evoluir x gerações, esse passo-a-passo deve ser executado x vezes. O Algoritmo termina quando é atingida a meta de aptidão por um número máximo de interações ou com o valor da aptidão de um cromossomo ultrapassando o mínimo aceitável, no entanto, o resultado é sempre um grupo de soluções mais próximas de um ótimo local ou um ótimo global.

Além de algoritmos genéticos, existem várias técnicas de inteligência computacional que também se inspiram na natureza, como as redes neurais ¹ e lógica nebulosa (fuzzy) ², porém, o objetivo final de todas as técnicas de inteligência computacional é a busca,

¹Em ciência da computação e campos relacionados, redes neurais artificiais (RNA) são modelos computacionais inspirados pelo sistema nervoso central de um animal (em particular o cérebro) que são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões.

²A lógica difusa é a forma de lógica multivalorada na qual os valores lógicos das variáveis podem ser qualquer número real entre 0 (FALSO) e 1 (VERDADEIRO). Diferentemente, na lógica booleana,

podendo ser uma solução numérica, do significado de uma expressão linguística, de uma previsão de carga ou de qualquer outro elemento que tenha significado em uma determinada circunstância.

De acordo com Linden (2012) não importa a área de aplicação, uma busca sempre tem como objetivo encontrar a melhor solução dentre todas as soluções possíveis (o espaço de soluções)³.

Os algoritmos genéticos são eficientes em varrer o espaço de soluções e encontrar respostas resoluções próximas da solução ótima, quase sem necessitar interferência humana.



Figura 7 – Fluxograma do Algoritmo Genético Sequencial

Fonte: acervo do autor

4.0.1 Algoritmos Genéticos Paralelos

Algoritmos Genéticos requerem muito tempo para serem executados, por isso, são bons candidatos para a paralelização. Entretanto, Linden (2012) afirma que mesmo chamado de paralelismo intrínseco, os algoritmos genéticos tradicionais não são paralisáveis,

os valores lógicos das variáveis podem ser apenas 0 e 1.

³O espaço de todas as soluções possíveis (que é um conjunto entre as quais a solução procurada está).

devido à sua inerente estrutura sequencial (avaliação > escolha dos pais > reprodução > população) e devido a necessidade de se utilizar um controle global sobre todos os indivíduos (realizados na população e seleção).

Com o tempo, foram propostas várias técnicas de paralelismo de algoritmos genéticos. Paz (1998) consideram as seguintes técnicas de se paralelizar um algoritmo genético:

- ❑ Panmitic;
- ❑ Granularidade Fina (Finely Grained);
- ❑ Granularidade Grossa (Island);
- ❑ Híbrida: Mistura entre Granularidade Fina e Granularidade Grossa.

Algoritmo Paralelo Panmitic A implementação de um Panmitic constitui uma classe mais simples de todas, sendo mais indicados para máquinas com memória compartilhada. Basicamente, este tipo de algoritmo genético consiste em outros vários algoritmos simples rodando em outros processadores, mais precisamente um em cada, e operando sobre uma mesma população global, conforme demonstrado na Figura 8.(LINDEN, 2012).

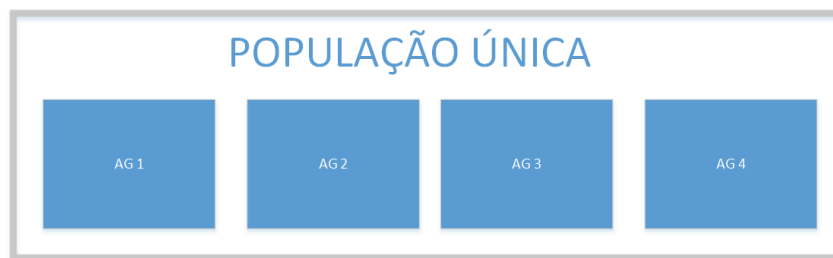


Figura 8 – Algoritmo Panmitic

Fonte: acervo do autor

No algoritmo Panmitic os processadores sincronizam na avaliação no operador de seleção, isto é, eles vão selecionando indivíduos e quando o número desejado já tiver sido escolhido, cada processador "expele" os filhos que gerou e que vão compor a nova população.

Este tipo de abordagem baseia-se no fato de que cada indivíduo pode ser avaliado sem conhecer o resto da população e cada operação genética pode ser realizada conhecendo-as apenas o(s) pai(s) selecionado(s). Assim, podemos realizar várias avaliações e várias operações genéticas (reprodução e mutação) simultaneamente, ganhando um grande tempo com a paralelização.

O problema é que, uma vez avaliados e operados, temos um conjunto de novos indivíduos e de pais previamente existentes, que são os candidatos a compor a nova população. Isto exige um único módulo de população capaz de decidir como será composta a nova

geração, decidindo quais pais serão descartados e quais filhos serão usados. Como esse processo é único, ele cria um gargalo no sistema não importando quantos processadores possuímos.

A complexidade de implementar esse algoritmo também pode ser uma desvantagem, sendo que exige muito mais tempo de desenvolvimento para sincronizar o paralelismo, que deve ter uma comunicação eficaz, para dizer quantos pais já selecionaram e quantos filhos sincronizaram, gerando assim uma grande sobrecarga de programação.

Esse esforço extra de desenvolvimento, pode ser resolvido com a solução mestre-escravo: Um processo que controla a seleção é criado para gerenciar qualquer paralelismo, sendo processador, thread ou processo, aliviará os indivíduos e quem realizará as operações genéticas sobre os cromossomos selecionados.

O processo mestre-escravo também pode trazer alguns problemas como:

- ❑ Dificuldade em manter o paralelismo balanceado. Podendo ter um balanceamento fixo ou dinâmico do número de indivíduos que é enviado para cada processador ou thread. Sendo o dinâmico mais recomendado para melhor resultado, mesmo que dependa de uma melhor comunicação entre os processos.
- ❑ O mestre pode ser um gargalo no sistema. A velocidade máxima do algoritmo é limitada pela capacidade do processador mestre em enviar e receber informações dos escravos.

A Figura 9 demonstra o fluxograma do algoritmo genético paralelizado em modelo panmítico, onde o processo decorre através de uma população, porém durante a avaliação de fitness é dividido entre as threads seguindo uma linha selecionar os pais e outra sequência normal do fluxo do algoritmo genético, passando para o cruzamento. O detalhe importante nesse fluxo é o processo de sincronização, que ocorre na primeira vertente, com a seleção de pais com o processo normal contínuo, depois da mutação, onde são expelidos os filhos e sincronizados com os pais para depois entrar no processo de seleção.

Com a particularidade do panmítico ter uma implementação mais indicada para problemas que utilizam máquinas com memória compartilhada ele não será utilizado no presente trabalho, sendo que a necessidade de melhoria do algoritmo escalável na busca de melhor rota na mineração depende de uma condição de paralelismo aplicado aos cores do processador trabalhando internamente as threads com o objetivo de reduzir o tempo de execução do algoritmo e também foi rejeitado pela maior complexidade em sua implementação.

Algoritmo Paralelo Granularidade Fina No algoritmo de granularidade fina os AG's são criados separados em cada processador, porém um indivíduo dessa população faz o cruzamento com um outro das populações vizinhas, sendo esse algoritmo também chamado de modelo de vizinhança.

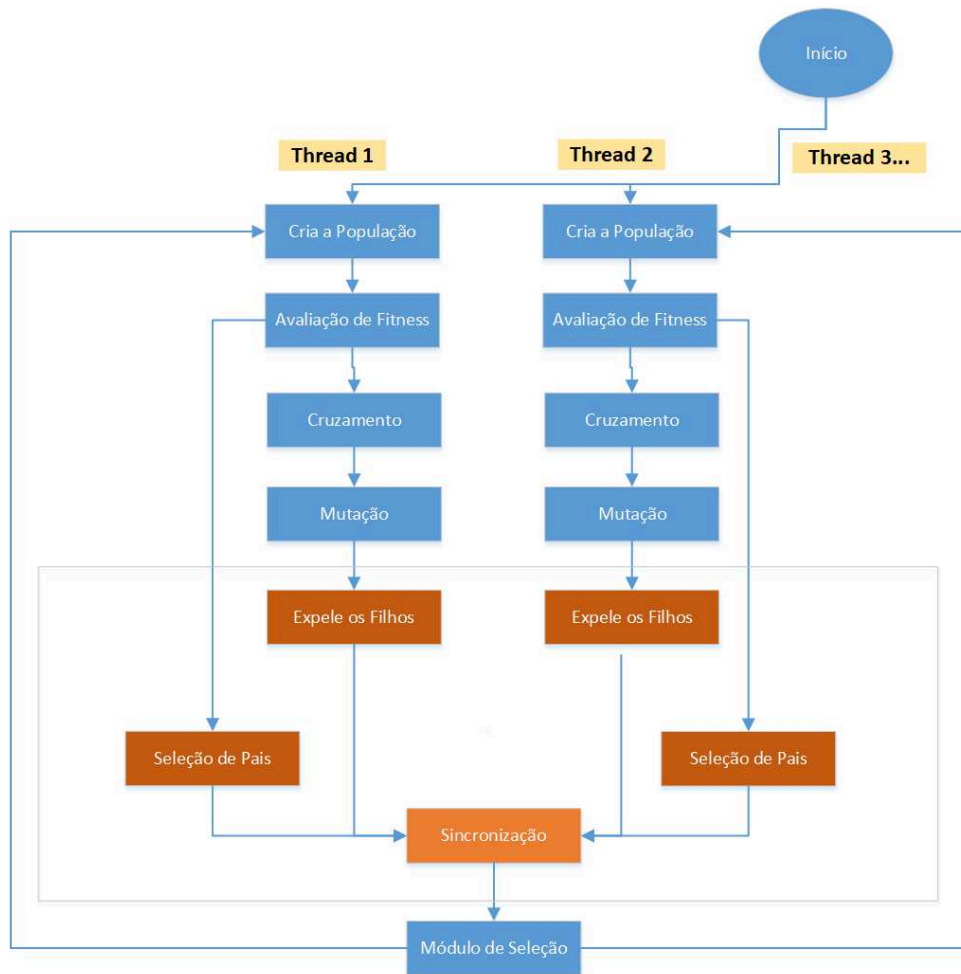


Figura 9 – Fluxograma do Algoritmo Panmitic

Fonte: acervo do autor

Esse algoritmo difere do Island sendo que ele evolui apenas uma popula o. Os indiv duos pertencentes a popula o em uma c lula de grade (ou hipergrade, dependendo de n dimens es) s o separados e os operadores gen ticos s o aplicados sobre os indiv duos vizinhos dessa grade. Essa troca de mensagens na troca de indiv duos pode deixar o sistema lento. (LINDEN, 2012)

Cada indiv duo pode ser alocado para um processador e interagir apenas com os outros que estiverem na sua vizinhan a, por m a topologia da grade deve ser definida corretamente antes.

O algoritmo de granularidade fina   extremamente adequado para computadores que tem uma estrutura massivamente paralela onde esse tipo de equipamento   voltado para atividades simples e locais. Logo um algoritmo gen tico aplicado nesse tipo de hardware deve ser para atividades simples que possam ser executadas simultaneamente.

V rios trabalhos apontam para o fato de que o desempenho do Algoritmo Gen tico paralelo piora quando aumenta de forma excessiva o tamanho da vizinhan a e comporta-se parecido com o Panmitic, estabelecendo um gargalo.

A Figura 10 demonstra o fluxo do algoritmo de granularidade fina, onde inicialmente são criadas as populações e inicia a sequência em cada thread. A característica importante desse algoritmo é após o fluxo de avaliação de aptidão, onde durante o cruzamento é criado uma grade, um vetor onde existe a troca de indivíduos, com cada indivíduo representando uma interação do paralelismo, caracterizando assim um processo de granularidade fina, conforme mostrado no capítulo e deste trabalho. Após essa troca o algoritmo segue conforme o modelo tradicional, passando para a mutação e iniciando novamente.

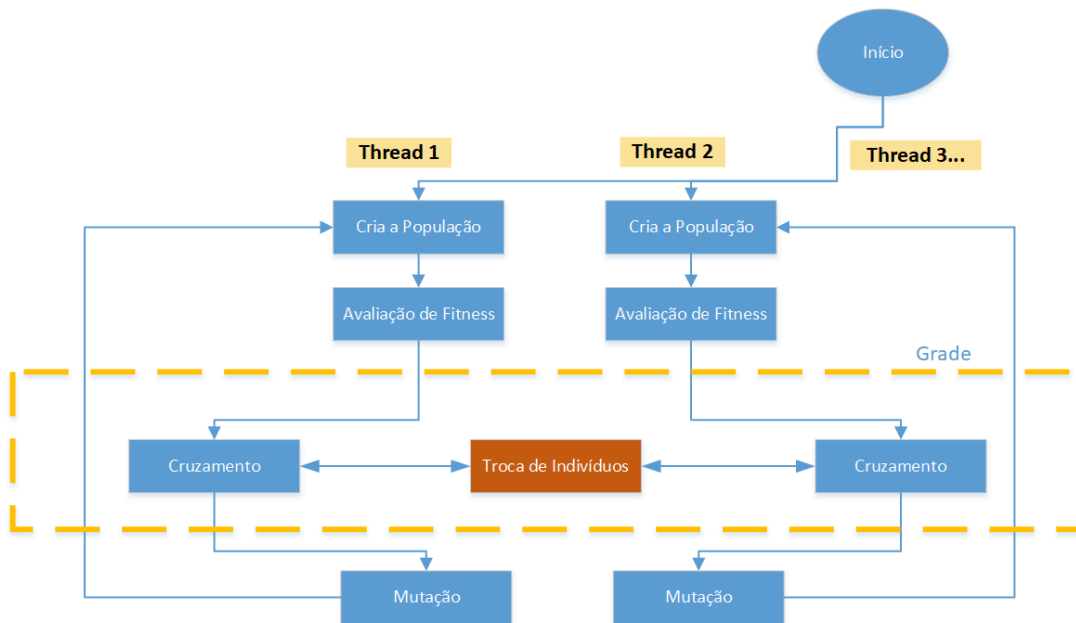


Figura 10 – Fluxograma do Algoritmo Granularidade Fina

Fonte: acervo do autor

O Algoritmo de granularidade fina não foi utilizado nessa proposta de trabalho de otimização de rotas porque possui um gargalo na troca de indivíduos entre a grade para o cruzamento, independentemente do número de processadores ele tende-se a ficar lento, isto é, exige elevado recurso computacional.

Para resolver esse problema, o sincronismo de indivíduos dentro da grade é necessário, o que gera um grande esforço de desenvolvimento e implementação, não sendo aplicável ao projeto.

Algoritmo Paralelo Island O Modelo de Ilhas (Island Model) consiste de uma proposta feita por Ochi et al. (1998), cuja premissa baseia-se no comportamento adaptativo que diferentes espécies demonstraram quando separadas em ilhas independentes, como por exemplo: aves em diferentes ilhas são diferentes umas das outras, embora possuam o mesmo antepassado.

Aplicando esta premissa computacionalmente, Ochi et al. (1998) implementou uma versão do AG, onde são criadas múltiplas populações independentes, cada qual desempe-

nhando as funções padrão do AG padrão.

Whitley, Rana e Heckendorn (1998) realizaram testes com e sem o operador de migração e perceberam que sem o operador as populações ainda assim conseguem obter boas soluções, porém ao aplicar o operador de migração a média das soluções é melhor, porém perdendo desempenho com o operador. Como a finalidade do trabalho consiste em utilização do modelo padrão do algoritmo de ilha, será implementado um operador de migração no algoritmo.

A Figura 11 demonstra o fluxo do algoritmo genético paralelo em modelo de ilha, considerado de granularidade grossa, onde tem interações apenas no início, quando a sequência cria uma população diferente em cada thread e no final executa um processo de migração, selecionando os melhores indivíduos para depois serem sincronizados após a evolução.

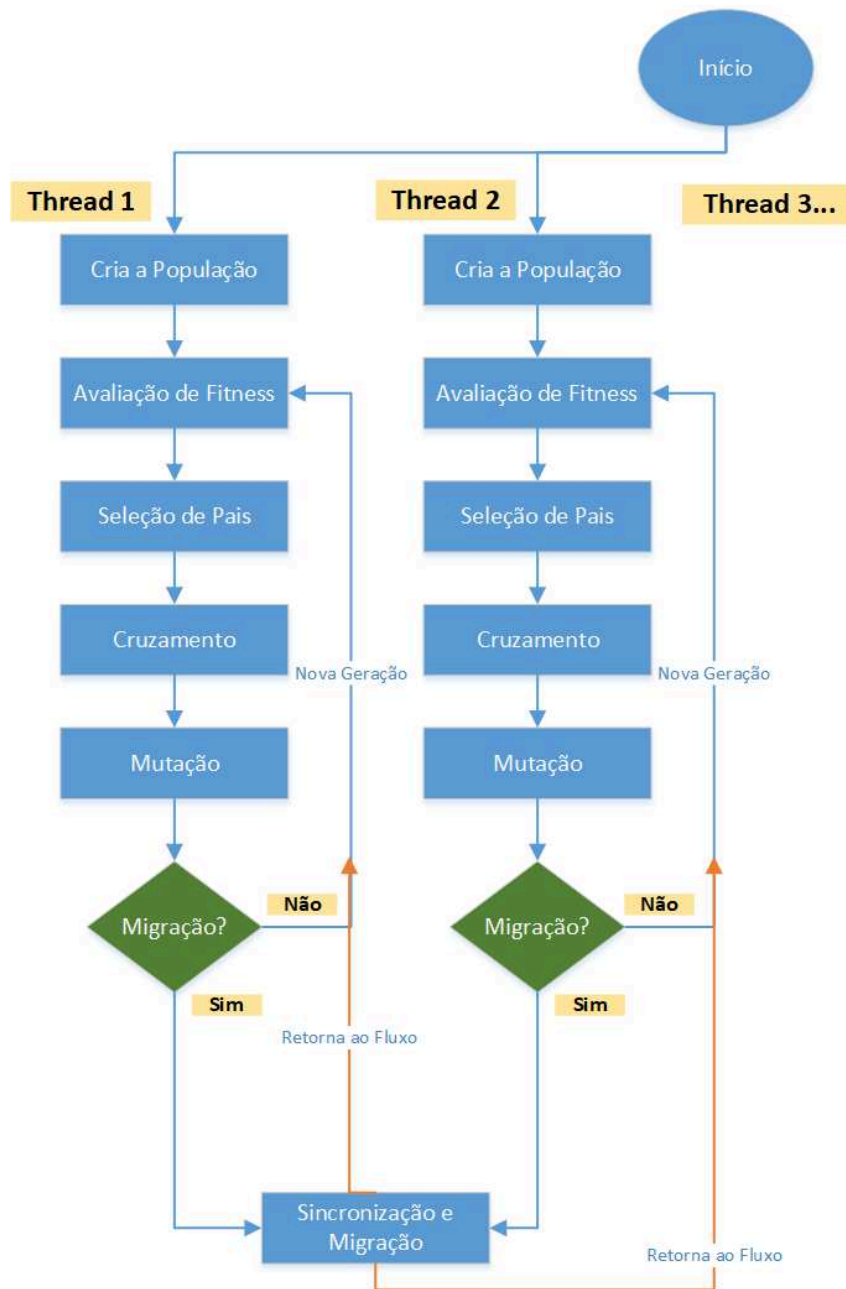


Figura 11 – Fluxograma do Algoritmo Island

Fonte: acervo do autor

Com o modelo de granularidade grossa (Island), a existência de um conjunto de subpopulações que são distribuídas entre os elementos do processamento é considerada evitando assim que as subpopulações se evoluam totalmente isoladas utilizando o mecanismo de migração.

A troca de indivíduos entre as subpopulações após uma determinada quantidade de gerações é chamada de migração e faz com que novos padrões devam ser estabelecidos antes da execução de um algoritmo genético, utilizando a granularidade grossa sendo eles o tamanho das subpopulações, controle da frequência das migrações, quais elementos

devem ser migrados e substituídos. (LINDEN, 2012).

Busca de Rotas

O Problema de Roteamento de Veículos, conhecido na literatura como Vehicle Routing Problem (VRP), tem sido um alvo de estudos nos últimos anos, principalmente na mineração, porque existem elevados custos envolvidos no processo de extração e transporte do material. A complexidade do problema, somada à necessidade de reduzir os custos que apresentam essas atividades tem estimulado diversos profissionais e pesquisadores a realizarem estudos teóricos sobre o tema.

Bengtsson Rastislav Galia e Kohl (2007) diz que a solução de roteamento de veículos se situa na classe dos problemas difíceis de serem resolvidos, ou seja, é um problema np-complexo ¹

Quando se tenta resolver problemas do tipo VRP na prática, encontram-se limitações para o tempo de entrega de resposta devido a complexidade em se encontra a solução do problema. Do ponto de vista da teoria da complexidade computacional, se não executar em tempo hábil, todos resultados podem ser ineficientes e não garantir o ganho da melhor rota escolhida para o equipamento naquele momento.

Conforme os pontos de destino, origens e cruzamentos aumentam, o tempo de execução do algoritmo cresce exponencialmente. O algoritmo de Dijkstra e o algoritmo Bellman-Ford são muito utilizados em problemas de VRP porque fornecem resposta rápida em comparação a algoritmos que utilizam Pesquisa Operacional (PO). No entanto, esses algoritmos são utilizados para resolver VRP que possuem poucas origens e destinos e poucos cruzamentos, desse modo, apresentando poucas combinações. (TORRES, 2017)

5.1 Problema na Mineração

O objetivo de utilizar a otimização de rotas na mineração com algoritmos genéticos neste trabalho é para obter a redução de custos de operação da mina, que vai desde a

¹NP-difícil (ou NP-hard, ou NP-complexo) na teoria da complexidade computacional, é uma classe de problemas que são informalmente considerados: “Pelo menos tão difíceis quanto os problemas mais difíceis em NP”.

extração do material até o britador.

Esta etapa representa um dos maiores custos da empresa, dos quais destacamos a manutenção e o combustível de carga e transporte de minério. (ALVARENGA, 1997)

A Figura 12 mostra a vista aérea de uma mina de grande porte, que nesse caso é considerada a maior mina de fostatato da américa latina. Pode observar pela imagem a complexidade de uma estrutura de operação de uma mina de grande porte.



Figura 12 – Mineração a Céu Aberto

Link: <http://www.defesaaereanaval.com.br/brasil-da-a-largada-na-corrída-por-metals-de-tablets-e-demisseis/> - Acessado em 11/11/2017

Antes de tentar otimizar a operação de frota na mineração, devemos antes entender suas características e conceitos. A mineração em céu aberto determina que o transporte de produto, do ponto de carregamento do material até o local de descarregamento é feito na superfície e que existem muitas possibilidades de rotas desse material transportado, principalmente no caso do trabalho proposto, sendo uma das maiores minas á céu aberto do Brasil e a maior de fostatato da América Latina.

Dentro da mineração existem dois tipos de materiais a serem extraídos e transportados: o minério e o material estéril, que não interessa para a maioria das empresas, sendo que a sua retirada é importante para se poder extrair o minério. A relação entre minério e material estéril consiste em um fator determinante do custo operacional. (ALVARENGA, 1997)

A figura 13 demonstra a operação de um carregamento de um equipamento de transporte, nesse caso um caminhão fora-de-estrada, por uma escavadeira, considerada uma

máquina de carregamento, que fica estática em um ponto de origem durante a extração do minério.



Figura 13 – Carregamento de Minério - Frente de Lavra

link: <http://www.ebah.com.br/content/ABAAABVtoAB/carregamento> - Acessado em 12/12/2017

Dentro da proposta desse trabalho o despacho computadorizado é executado dentro de um sistema embarcado e automaticamente envia os equipamentos de transporte para outras rotas. A tomada de decisão é gerada por um algoritmo que é alimentado com as variáveis que são coletadas pela rede naquele momento. Se o algoritmo demorar a executar e, conseqüentemente, tardar a fornecer uma resposta, toda a logística da mina e posicionamento de veículos de transporte podem ser afetados, diminuindo assim a eficiência do transporte do minério.

Os locais de retirada de material são chamados de frentes de lavra ou áreas de escavação e tem equipamentos de carga, como escavadeiras e carregadeiras, que são utilizados para carregar os caminhões com uma quantidade padronizada de material, dependendo da capacidade dos equipamentos de transporte e das regras da mineradora, os quais transportam-no através de uma das rotas existentes até o ponto de basculamento, que geralmente pode ser um britador. Os equipamentos de transporte realizam ciclos de carregamento e descarregamento repetidamente, e é comum dois ou mais caminhões se encontrarem em um dos pontos de carregamento, nas frentes de lavra ou nos de descarregamento, no britador. Essas são as situações que devem ser evitadas, pois são consideradas perda de produção e gera prejuízos para a área de operação de mina. (TORRES, 2017)

A Figura 14 demonstra em uma visão horizontal a sequência de um ciclo de produção, entre a origem em uma frente de trabalho, onde fica localizada a máquina de carga até o ponto de destino, que é representado pelo britador. Nesta imagem mostra a ocorrência de filas dos equipamentos de transporte ao aguardar a máquina de carga, que é considerada perda de produtividade em uma operação de mina.

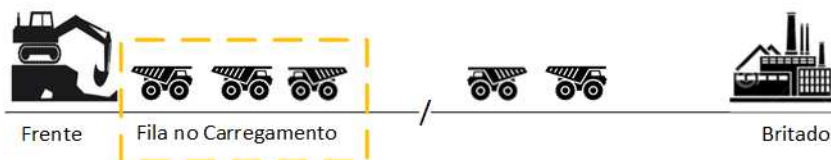


Figura 14 – Ciclo de Transporte com Fila no Carregamento

Fonte: Acervo do Autor

Quando um sistema de despacho é computadorizado, ele sugere automaticamente novas rotas ao equipamento de transporte, após um gatilho que deve ser executado nos seguintes momentos: fim de carregamento, fim de descarregamento e dentro de cruzamento entre rotas.

Considerando que uma área de cruzamento entre duas ou mais rotas seja de 30 metros de raio e que os caminhões deslocam-se em um padrão médio de velocidade de 35 Km/h, o tempo que se tem para informar ao motorista a rota que ele deve seguir é de no máximo 3 segundos. Deve-se considerar também o tempo de percepção do motorista para observar a mensagem do sistema de despacho para a alteração de rota, que também é emitido um alerta sonoro para o motorista, poder fazer a troca de rota em tempo hábil.

Um GPS, presente no sistema de despacho do caminhão é utilizado para fornecer a latitude e longitude do local em que ele se encontra com o objetivo de se criar uma cerca eletrônica.² O polígono da cerca eletrônica pode ser implementado por meio de formas geométricas simples como: quadrado, círculo, polígono, ou formas geométricas aleatórias compostas por diversas coordenadas.

Quando um caminhão entra em uma cerca eletrônica, deve receber a informação de alteração de rota e pode, assim, tomar uma melhor decisão. A maioria dos sistemas computadorizados de despacho contam com uma tela touch screen e um buzzer que emite um sinal sonoro de mensagens, para conseguir a atenção do motorista. Além disso o equipamento também é posicionado sempre a vista do motorista, que pode visualizar e escutar a mensagem de alteração de rota gerada pelo algoritmo genético no sistema embarcado e tomar a ação.

A figura 15 demonstra o exato momento de uma possível tomada de decisão de um equipamento de transporte na velocidade média de 35 km/h, ao chegar ao ponto de tomada de decisão de alteração de rota. Neste momento o equipamento embarcado deve

²A cerca eletrônica corresponde a marcação de uma área através de um sistema de geoprocessamento que faz parte de um sistema de despacho.

executar o algoritmo genético e retornar a mensagem em tempo hábil para a tomada de decisão do motorista.

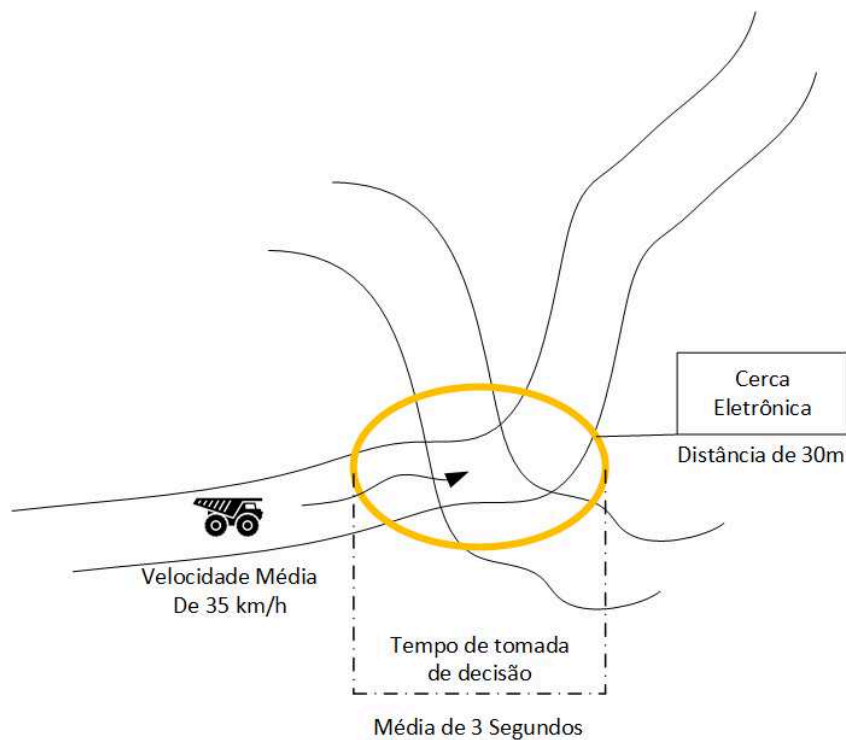


Figura 15 – Cerca Eletrônica na Tomada de Decisão

Fonte: Acervo do Autor

Em uma mineradora de pequeno porte com até 20 equipamentos (TORRES, 2017) demonstrou com a sua solução que embarcando o algoritmo genético, de forma sequencial, a tomada de decisão da melhor rota garantindo assim uma menor perda de tempo com filas e aumentando a produtividade.

Torres (2017) converte o mapa da mina em um grafo, onde cada ponto de lavra, cruzamento, britagem e de alimentação representam um nó. Entre os nós é atribuída a propriedade da rota, por exemplo a distância que o caminhão deve percorrer para sair de um ponto e chegar ao outro. Cada nó recebe um index iniciado com um até o número máximo de nós e representarão os genes do cromossomo. Cada cromossomo é formado pelo agrupamento sequencial dos genes de acordo com as rotas que ligam a origem ao destino. Isso resulta em cromossomos de tamanho diferentes, dependendo da rota traçada.

A Figura 16 mostra um diagrama de uma mina de pequeno porte, demonstrando as opções de cruzamentos para várias possibilidades entre origens, onde estão as máquinas de cargas, como escavadeiras, e destinos, que possivelmente podem ser britadores em indústrias.

Escalabilidade do Algoritmo de Decisão de Rotas na Mineração Normalmente em problemas de busca de rotas, a aptidão considera a menor distância como o

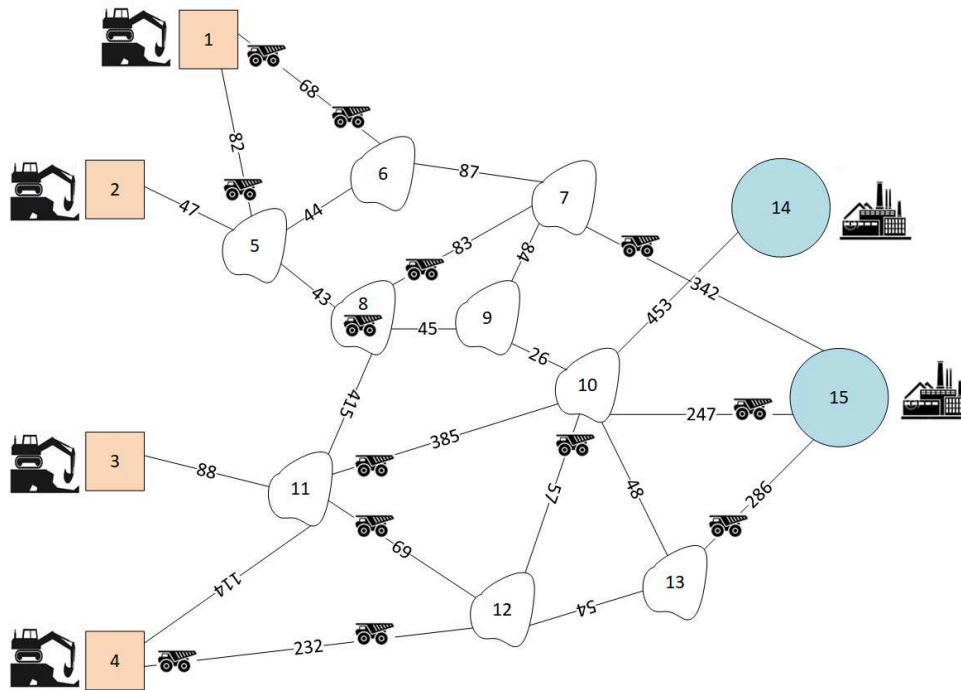


Figura 16 – Diagrama de Mapeamento de Rotas de Ponto Origem e Destino

Fonte: Torres (2017)

objetivo principal para o transporte do material a ser processado. Contudo, o problema pode levar em conta outros fatores, como trânsito, condições das vias, hábitos comuns dos motoristas, pontos obrigatórios, qualidade do material que está sendo transportado, economia de combustível etc. Em problemas de alocação dinâmica, a função de aptidão leva em consideração os seguintes fatores: Minimizar a Distância Média de Transporte, diminuir a formação de filas e o reduzir a ociosidade das máquinas. (TORRES, 2017)

Uma mina de grande porte, conhecida como a maior mina de fóstato da América Latina foi base de utilização de dados de padrões das variáveis necessárias para modular a função objetivo com os parâmetros reais cadastrados de um sistema de despacho utilizado na mesma. Esses parâmetros tem limitação de dados dentre as características da mina, como origens, destinos, DMT e todas as outras variáveis necessárias para a execução do algoritmo.

A Figura 17 mostra a visão de um sistema de despacho para os equipamento em uma mineradora de grande porte. Nessa visão consegue-se ver a complexidade de uma operação de mina com várias opções de rotas e números de equipamentos.

Dentre os padrões da mina, consegue-se escalonar o algoritmo dentre todas as opções possíveis com dados reais, testando assim todos as combinações de rotas possíveis e números de equipamentos.

Essa escalabilidade foi prevista para podermos realizar os testes desde uma mineradora de pequeno porte, conforme Torres (2017), e evoluir até uma de médio porte ou até mesmo uma de grande porte, conforme os dados utilizados na pesquisa.

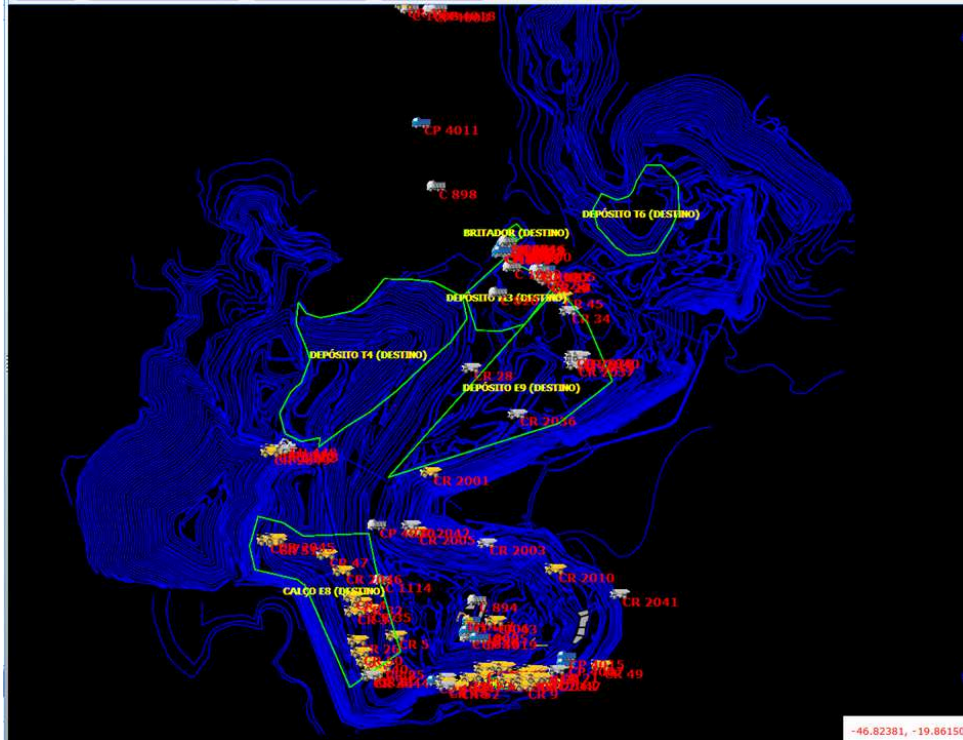


Figura 17 – Mapa Real da Mina de Fostato

Fonte: Torres (2017)

Uma mina de pequeno porte pode apresentar um número de 20 equipamentos, 2 destinos, 4 origens e 8 cruzamentos, dentro de um diâmetro da mina de até 2km de extensão. Porém nos dados da mineradora que pesquisados como padrão máximo da escalabilidade, esses dados podem chegar até 170 caminhões e várias origens, destinos e cruzamentos.

Dentro de um padrão escalável de complexidade, o comportamento do algoritmo genético tende a exigir mais de processamento e hardware sendo que primeiramente o indivíduo, formado conforme o trabalho proposta de Torres (2017), e utilizado nesse projeto, aumenta cada vez mais o seu tamanho dependendo de um maior número de possibilidades para chegar da origem para o destino, conforme a Figura 18.

Contudo, o número de equipamentos tomando a decisão no momento, selecionando a melhor rota após a execução do algoritmo, tende a exigir mais ainda do hardware, aumentando assim a sua população.

Com esses parâmetros de escalabilidade, este trabalho apresenta a solução para a execução do algoritmo genético considerando minas de pequeno, médio e grande porte.

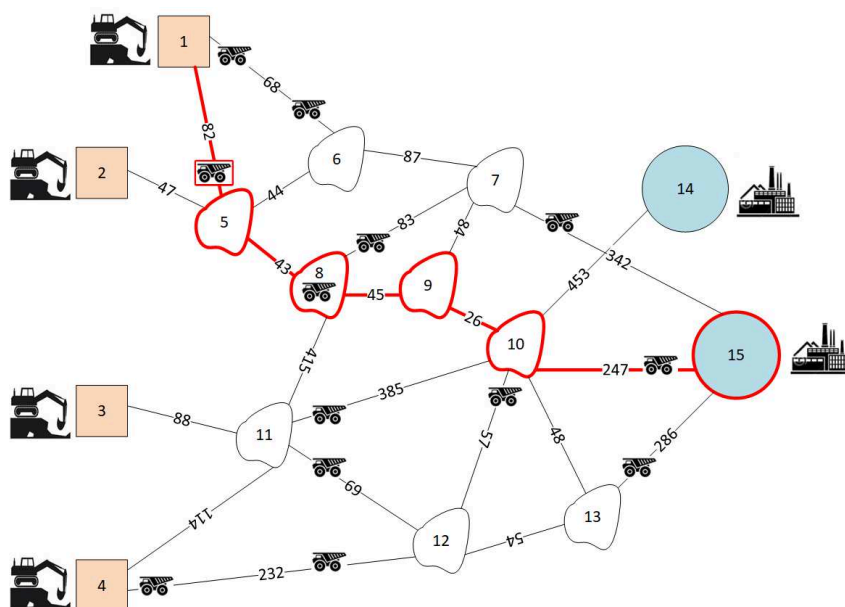


Figura 18 – Exemplo de Melhor Rota

Fonte: Acervo do Autor

Metodologia

A metodologia desse trabalho tem o objetivo de desenvolver um sistema auxiliar, com hardware e software, para executar o algoritmo genético paralelo e ser possível a utilização em um sistema de despacho existente em uma mineradora de grande porte. Esse algoritmo captura a leitura dos parâmetros para garantir a execução do algoritmo genético e obter a melhor rota para o equipamento, garantindo assim melhor produtividade.

Esta proposta é diferente do trabalho de Torres (2017) onde foi desenvolvido uma interface de um novo sistema de front-end ¹ apenas para a interação da resposta do algoritmo genético embarcado.

A Figura 19 demonstra o sistema de front-end desenvolvido por Torres (2017) para enviar a mensagem para o operador de mudança de rota após receber o resultado do algoritmo genético.

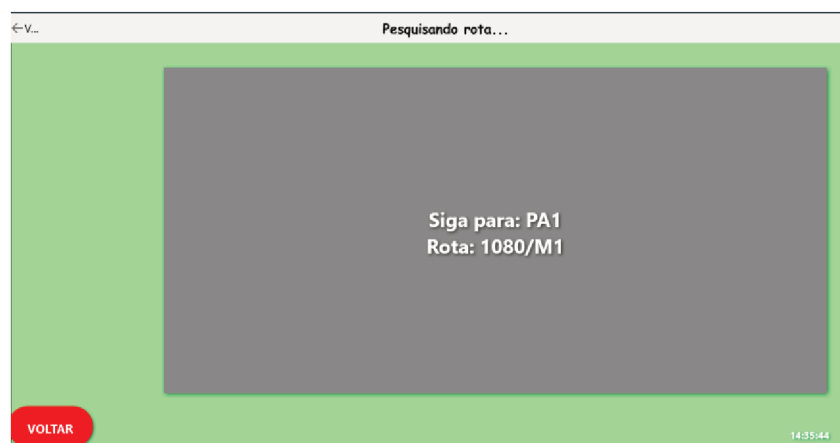


Figura 19 Imagem da interface da solução proposta por Torres (2017)

Fonte: Torres (2017)

Será feito também um comparativo dos hardwares atuais utilizados em pesquisas para a análise de melhores resultados em sistemas embarcados com algumas opções de recursos

¹O front-end é responsável por coletar a entrada do usuário em várias formas e processá-la.

de processamentos paralelos com multithread, conforme demonstrado tendo o objetivo do algoritmo genético embarcado de executar dentro do momento de travessia da cerca eletrônica, onde são os pontos de cruzamentos de rotas, ou após os gatilhos de carregamento ou descarregamento, ocorrendo no final de cada evento, respectivamente, sugerindo uma nova rota para o motorista.

A limitação se dá também sendo que o número de equipamentos e rotas que podem ser utilizados para gerenciamento na mineradora e fica restringido a eficiência da análise dos dados e cálculo da otimização da melhor rota, conforme Almasi e Gottlieb (1989), que neste caso utiliza-se algoritmo genético. O desempenho do algoritmo de busca de rotas embarcado pode ser melhorado com o gerenciamento das threads, utilizando as técnicas de paralelismo considerada nesta proposta de trabalho, e trazer assim uma resposta mais rápida para o equipamento que aguarda o seu direcionamento para uma nova origem ou destino aumentando sua produtividade e evitando perdas.

A Figura 20 mostra um sistema de despacho existente em uma mineradora enviando a mensagem de alteração de rota para um motorista.

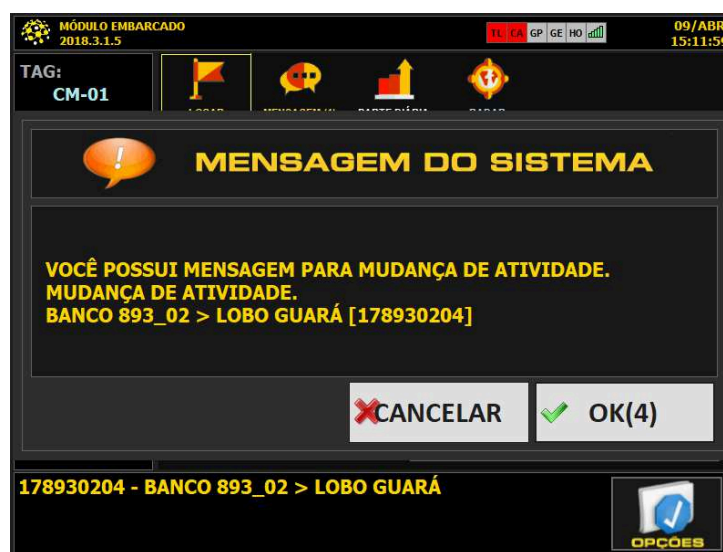


Figura 20 – Sistema embarcado no equipamento recebendo mensagem de alteração de rota.

Fonte: Acervo do Autor

Mesmo com a solução dentro dos equipamentos embarcados o processamento passa a ser muito requisitado e o sistema operacional não consegue efetuar com dinâmica todos os pacotes de dados ou métodos do sistema em ordem correta e causa necessidade do hardware embarcado ser mais robusto e gerando maiores custos.

Isso torna necessário uma melhor utilização dos recursos do hardware existente, sendo que nesse trabalho é utilizado o paralelismo do algoritmo genético com o modelo de ilha com o OpenMP, sendo que ambos foram escolhidos por ter portabilidade e fácil gerenciamento dos recursos de paralelismo. Adicionalmente, o ganho de desempenho do

paralelismo do algoritmo genético para obter melhor rota poderá minimizar custos de investimentos de hardware e software.

O ambiente de execução do software embarcado e do algoritmo genético a ser paralelizado nos equipamentos compõe de um conjunto de hardwares embarcados ARM com Raspberry Pi 2 com 4 núcleos, ARM com Raspberry Pi 3 com 4 núcleos, ARM com Banana Pi com 8 Núcleos.

As Figuras 21, 22 e 23 mostram respectivamente as imagens dos hardwares utilizados neste projeto.



Figura 21 – Raspberry PI 2

link: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/> - Acessado em: 01/02/2018



Figura 22 – Raspberry PI 3

Link: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> - Acessado em: 01/02/2018

A disponibilidade da rede de comunicação deve ser em tempo real, que garante a comunicação das unidades móveis com os computadores embarcados com um servidor central, onde é feita a coleta e processamento de dados. O Sistema embarcado com o hardware auxiliar para a execução do algoritmo genético acessa esse banco de dados e captura as variáveis necessárias para executar e trazer o retorno em forma de mensagem para o motorista.

Os usuários através de um software gerencial acessam esses dados em formato de relatórios e telas de cadastro e consulta em diferentes módulos do sistema como: planejamento, operação, manutenção, produção e outros.



Figura 23 – Banana PI

link: <http://www.banana-pi.org/m3.html> - Acessado em: 01/02/2018

O sistema operacional do sistema embarcado auxiliar, onde é executado o algoritmo, é de plataforma linux, compilado para atender as necessidades básicas do sistema em ambos equipamentos. Esse sistema não teve nenhuma compilação especial, sendo que os processos executados como padrão são determinados pelas versões de seus fabricantes, sendo elas:

- ❑ Raspberry Pi2 = RASPBIAN STRETCH WITH DESKTOP
- ❑ Raspberry Pi3 = RASPBIAN STRETCH WITH DESKTOP
- ❑ Banana Pi M3 = UBUNTU 16.04 MATE

Os processos executados em um sistema operacional padrão do fabricante extraído com a ferramenta HTOP ², do Linux e demonstrado conforme as imagens.

A Figura 24 demonstra os processos de um sistema operacional linux com a ferramenta HTOP. Nesta ferramenta são exibidos todos os processos que estão sendo executados em tempo real do sistema operacional.

6.0.1 Escalabilidade

A escalabilidade do algoritmo genético, que dentro do conceito da aplicação utilizada tem o seu individuo incrementado de acordo com a configuração da mina, condiz que quanto mais rotas possíveis com opções de cruzamento entre a origem e destino, tende a aumentar o tempo de execução.

Caracteriza-se nesse trabalho as seguintes configurações dos portes das mineradoras utilizadas:

²O HTOP é uma ferramenta para monitorar e gerenciar redes, além de ter muitos recursos que demonstram tudo através de gráficos e informações detalhadas que permitem com que haja interação entre usuários

Mina de pequeno porte, que é caracterizada como de 5 a 40 equipamentos, 4 origens de carregamento de carga, 2 destinos e 8 cruzamentos.

Mina de médio porte, com características de variáveis como de 1 a 110 equipamentos, 6 origens de carregamento de carga, 3 destinos e 32 cruzamentos.

Mina de grande porte, com uma configuração de 8 pontos de origem para carregamento, 62 cruzamentos de opções de rotas e com 5 opções de destino com uma variação de 1 a 170.

Para medir a escalabilidade de número de equipamentos do algoritmo genético, um sistema auxiliar em shell script³ foi desenvolvido, cujo apenas executa o programa principal do algoritmo genético com dados e variáveis escaláveis. Dessa forma, a cada vez que o sistema escalável é executado, ele adiciona uma linha em um log, no formato de Comma-separated values (CSV)⁴, os dados das variáveis escaláveis que foram incrementados, que neste caso é o número de caminhões, e o tempo resposta da execução, conforme modelo abaixo.

```
1
2 #!/ bin / bash
3 x=1
4 while [ $x -le 110 ]
5 do
6 #echo "Welcome_ $x_ times"
7 ./SearchPathGA 0 32 $x
8 echo - $x Equipamentos
9
10 x=$(( $x + 1 ))
11
12 done
```

O tempo medido da execução do algoritmo genético é captado pelo comando TIMER, que retorna a sua execução em milissegundos (ms), conforme demonstrado no código abaixo:

```
1 ...
2 result= truck.theBest ();
3
4 minaDinamica.mainFunction(result , false );
5
6 //Adiciona a linha do retorno da função do timer;
```

³Shell script é uma linguagem de script usada em vários sistemas operativos (operacionais), com diferentes dialetos, dependendo do interpretador de comandos utilizado. Um exemplo de interpretador de comandos é o bash, usado na grande maioria das distribuições GNU/Linux.

⁴O CSV é um implementação particular de arquivos de texto separados por um delimitador

```
7 AddLog(minaDinamica.ncm,(float( clock () - begin_time )*1000.0 /  
CLOCKS_PER_SEC) );  
8  
9 std::cout << (float( clock () - begin_time )*1000.0 / CLOCKS_PER_SEC);  
10 //referencia do timer;  
11  
12 }catch (const std::exception& e) {  
13 std::cout <<"ERROR:␣"<< e.what();  
14  
15 }  
16 ...
```

O algoritmo é executado todas as combinações das variáveis, como origens, destinos, número de equipamentos são testadas e escaladas.

Resultados

O algoritmo genético proposto deve ser executado de modo que o motorista possua tempo suficiente para alterar a rota quando entrar em uma cerca eletrônica. Lembrando que a cerca é um círculo virtual de aproximadamente 30 metros de raio ao redor de um cruzamento e que o caminhão trafega a aproximadamente 35 km/h, o algoritmo genético precisa ser executado em um tempo máximo de 1 segundo para que o motorista possua tempo suficiente para mudar a sua rota.

O objetivo principal desse projeto é melhorar o desempenho do algoritmo genético proposto por meio de computação paralela. A técnica que foi utilizada é a de Ilha (Island) executando a função de migração, conforme proposto por Linden (2012).

Para se obter os resultados experimentais, os hardwares utilizados utilizaram a versão de sistema operacional disponibilizada pelo fabricante. Os sistemas operacionais não foram otimizados.

Os testes foram realizados nos hardwares: banana Pi (possui 8 núcleos), Raspberry Pi 3 (possui 4 núcleos; 64 Bit; 1,2 GHz) e Raspberry Pi 2 (possui 4 núcleos; 32 Bit; 1,2 GHz) respectivamente.

Em cada teste, primeiramente, o algoritmo proposto é executado de modo sequencial, isto é, usando apenas uma thread e na sequência de modo paralelo em 2, 4 e até 8 núcleos, caso disponível, sendo que cada thread criada será executada em um núcleo diferente.

Os hardwares utilizados foram alimentados por meio de fonte de alimentação com saída de 5 volts e 2 amperes para todos os testes efetuados em todos os hardwares. Os testes foram realizados em laboratórios.

Uma das características que pode exercer forte influência nos resultados demonstrados é a arquitetura, número de núcleos e frequência de clock da CPU.

A Tabela 1 mostra as especificações técnicas do hardware Banana Pi, que apresenta um processador octa-core.

Tabela 1 – Especificação Banana Pi M3

Fonte: <http://www.banana-pi.org/m3.html>

Descrição	Banana Pi M3
CPU	A83T ARM Cortex-A7 octa-core, 1.2GHz, 512 KB L1 cache 1 MB L2 cache
Memória	2GB LPDDR3 (shared with GPU)
Wireless	802.11 b/g/n (AP6212)
GPIO	40 Pins: GPIO, UART, I2C bus, I2S bus, SPI bus, PWN, +3.3v, +5v, ground
USB	2x USB 2.0, USB OTG(Micro USB)
SD	Micro SD

A Figura 24 mostra que a paralelização do algoritmo genético para uma mina de pequeno porte não foi relevante em termos de desempenho do processamento do algoritmo. O hardware Banana Pi, quando executou 8 threads, uma em cada núcleo, apresentou melhor desempenho com poucos equipamentos e, a medida que o número de equipamentos foi crescendo, o desempenho diminuiu. Para o número máximo de equipamentos, a paralelização utilizando 2, 4 e 8 núcleos não foi relevante em termos de tempo de execução do algoritmo. Observa-se no gráfico que o tempo de execução do software começa mais rápida com menor número de equipamentos em todas as execuções, portanto é mais rápida ainda no início com a execução utilizando 8 Threads que demonstrado nas barras de cor cinza e aproximando das outras execuções ao final da escalabilidade em número dos equipamentos.

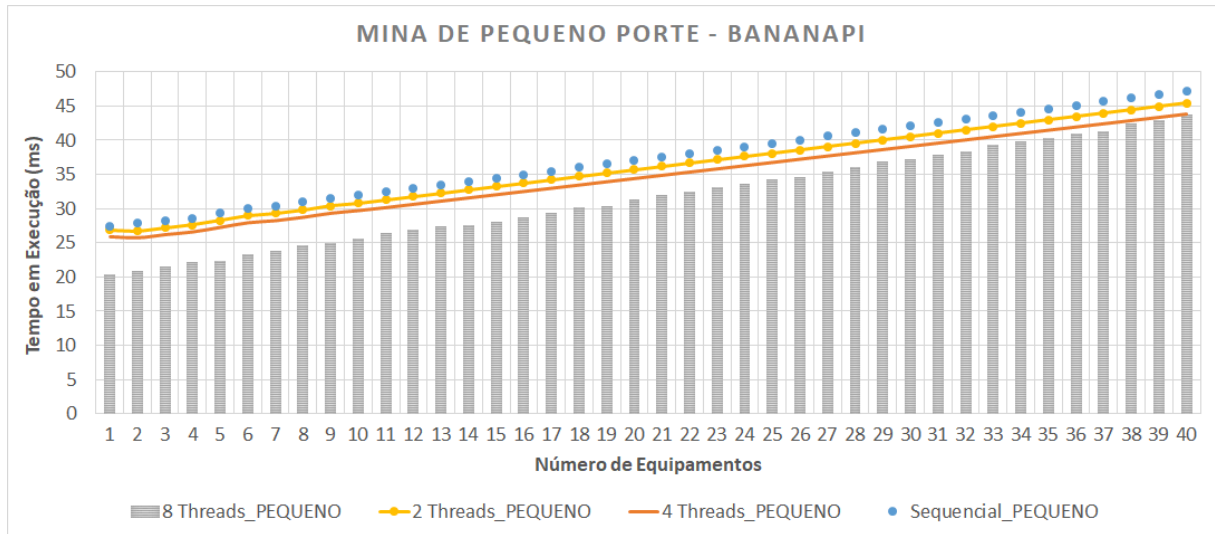


Figura 24 – Mina de Pequeno Porte - Banana Pi
 Fonte: Acervo do Autor

A Figura 25 mostra que a paralelização do algoritmo genético para uma mina de médio porte. Nesse caso, os resultados foram mais relevantes em termos de redução de tempo de execução do algoritmo, pois a medida que se utilizou mais threads, obteve-se ganho em desempenho.

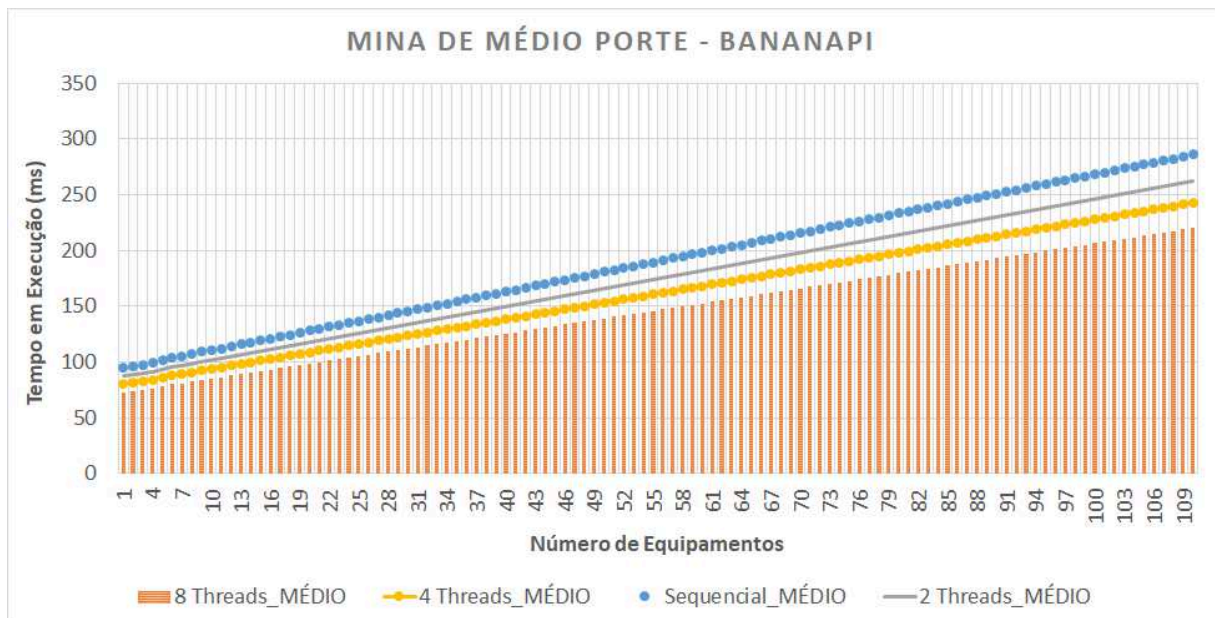


Figura 25 – Mina de Médio Porte - Banana Pi
 Fonte: Acervo do Autor

A Figura 26 apresenta os resultados para uma mina de grande porte. O tempo de execução aumenta de acordo com o número de equipamentos. Por outro lado, o paralelismo proporciona ganho de eficiência à medida que o número de threads e núcleos utilizados aumenta e de acordo com a quantidade de equipamentos utilizados na mina.

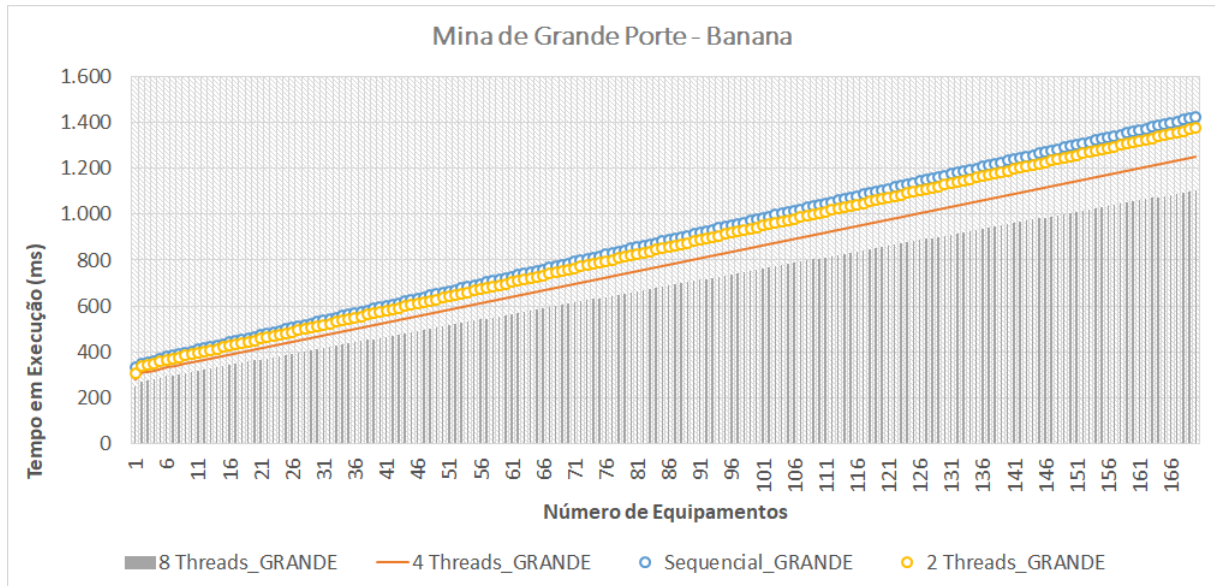


Figura 26 – Mina de Grande Porte - Banana Pi
 Fonte: Acervo do Autor

A Tabela 2 mostra as especificações técnicas do hardware Raspberry Pi2:

Tabela 2 – Especificação Raspberry Pi2

Fonte: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>

Descrição	Raspberry Pi2
CPU	A 900MHz quad-core ARM Cortex-A7 CPU
Memória	1GB RAM
Wireless	NO
GPIO	40 GPIO pins
USB	4 USB ports
SD	Micro SD card slot

A Figura 27 mostra os resultados de uma mina de pequeno porte. Observa-se que o paralelismo do algoritmo não proporciona relevância de ganho em eficiência de tempo de execução em função do número de equipamentos nem com relação a quantidade de threads utilizadas.

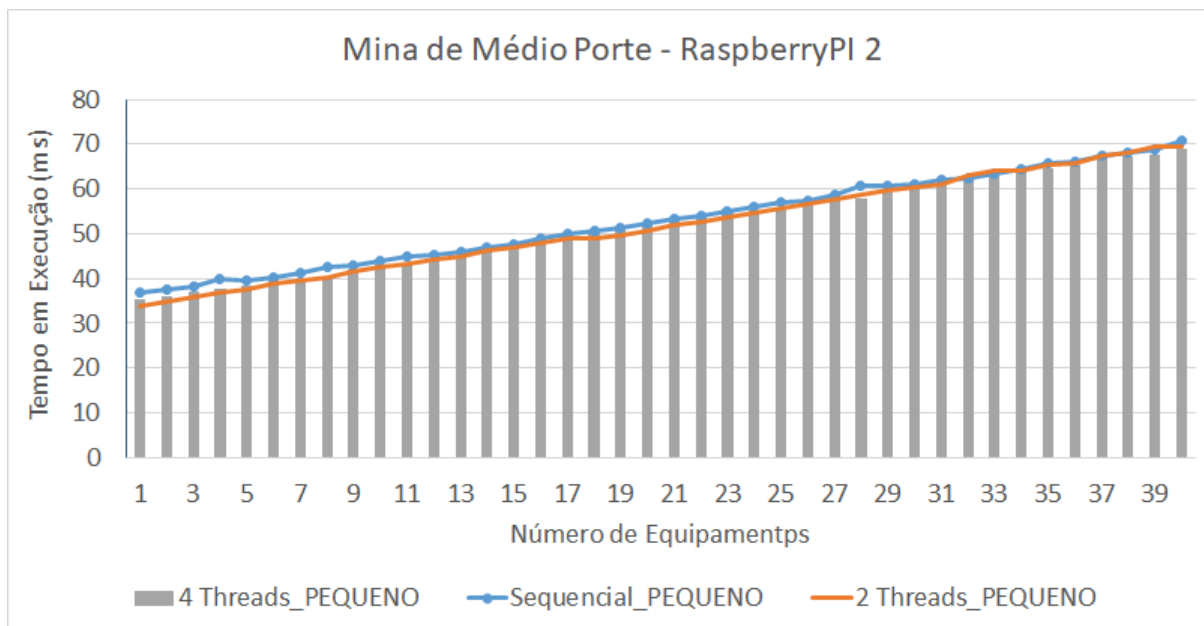


Figura 27 – Mina de Pequeno Porte - Raspberry Pi2
 Fonte: Acervo do Autor

A Figura 28 mostra os resultados de uma mina de médio porte com a execução no hardware RaspberryPi 2. Nota-se que os testes realizados com 4 threads foram melhores do que aqueles feitos com apenas duas. Nesse teste também se obteve melhor desempenho para um maior número de threads a medida que mais equipamentos são acrescentados.

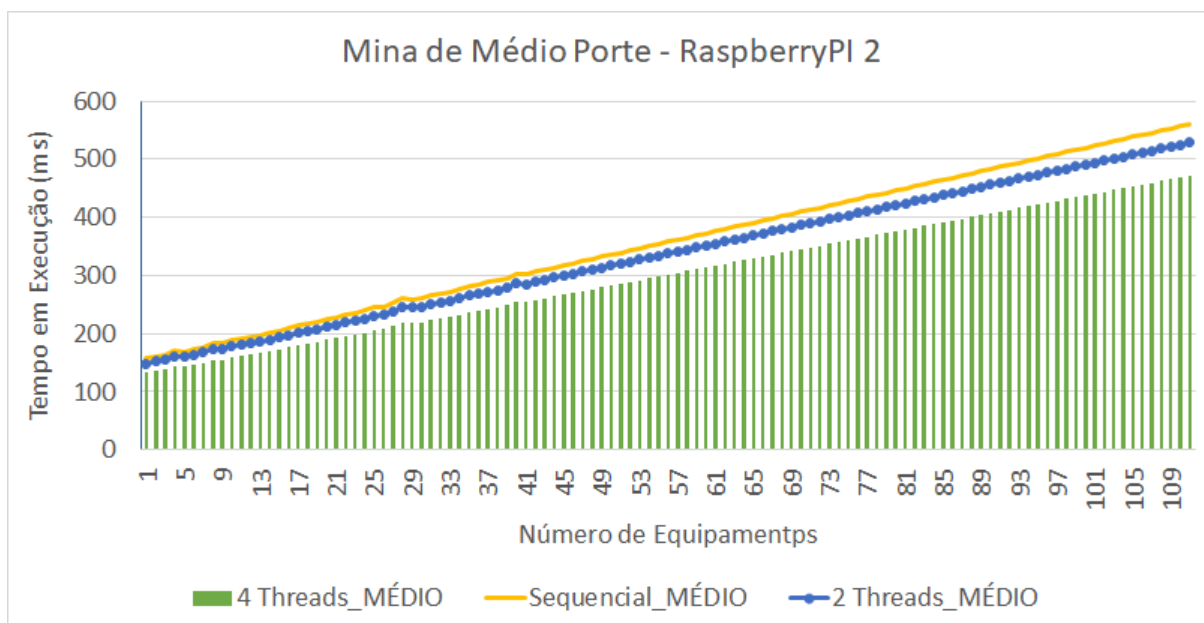


Figura 28 – Mina de Médio Porte - Raspberry Pi2
 Fonte: Acervo do Autor

A Figura 29 demonstra o desempenho do Raspberry Pi 2 para o cálculo do algoritmo genético proposto para uma mina de grande porte. Nesse ensaio, o tempo de execução do algoritmo genético ultrapassou 1 segundo que era o tempo limite de processamento. Isso

significa que não sobrar tempo suficiente para o motorista reagir e mudar de rota, caso necessário. Para uma mina dessa magnitude, deve-se optar por um hardware com maior poder de processamento.

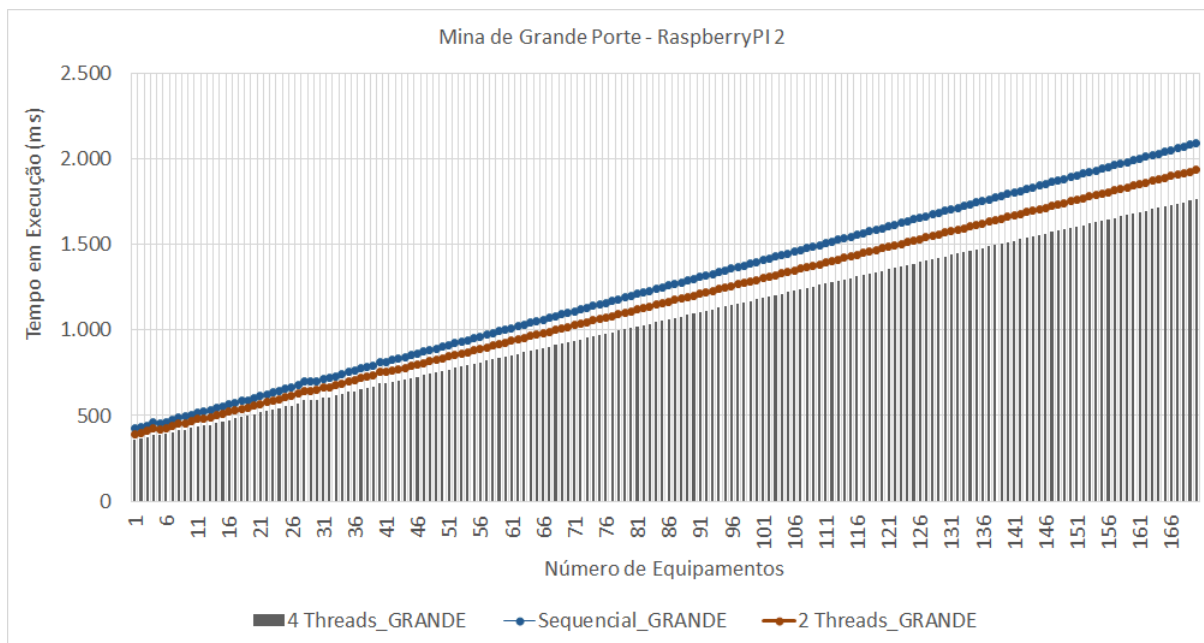


Figura 29 – Mina de Grande Porte - RaspberryPi2

Fonte: Acervo do Autor

A Tabela 3 mostra as especificações técnicas do hardware Raspberry Pi3:

Tabela 3 – Especificação Raspberry Pi3

Fonte: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

Descrição	Raspberry Pi3
CPU	1Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Memória	1GB RAM
Wireless	BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
GPIO	40-pin extended GPIO
USB	4 USB 2 ports
SD	Micro SD port for loading your operating system and storing data

A Raspberry Pi3 e Raspberry Pi2 possuem quatro núcleos, no entanto, a Raspberry Pi2 possui clock de processamento de 900Mhz e a Pi3 de 1,2Ghz. A seguir é apresentado os resultados experimentais obtidos com a Raspberry Pi3.

A Figura 30 mostra o resultado para uma mina de pequeno porte. Nota-se que, mesmo para uma mina pequena, consegue-se um desempenho de execução diferenciado quando se utiliza 4 núcleos e a medida que o número de equipamentos aumenta.

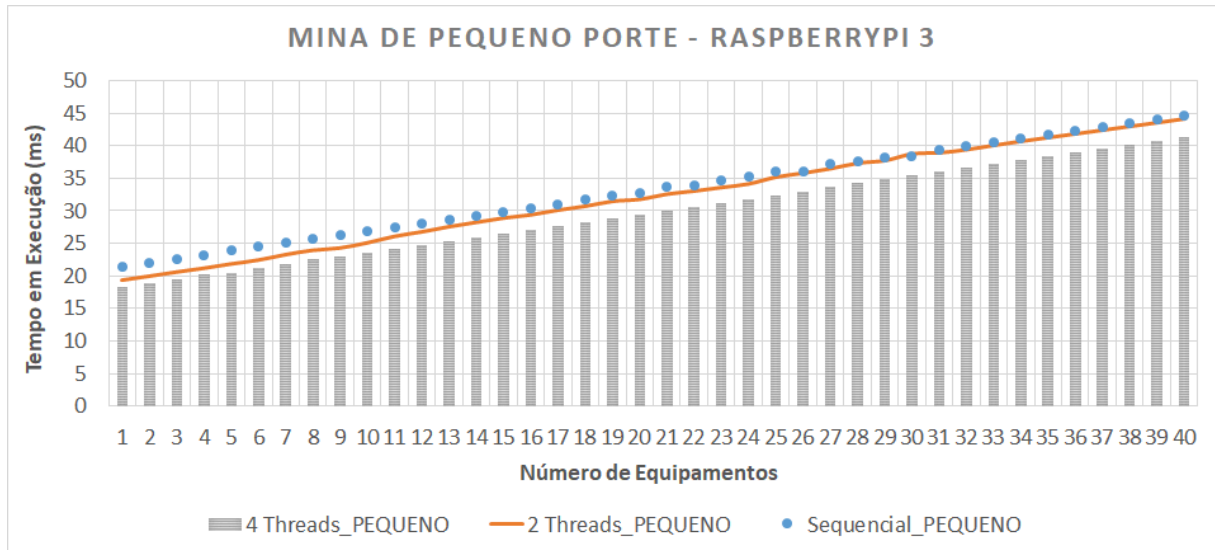


Figura 30 – Mina de Pequeno Porte - Raspberry Pi3
 Fonte: Acervo do Autor

A Figura 31 mostra o resultado na mina de médio porte. Nota-se que o ganho de desempenho se observa que o desempenho melhora conforme o indivíduo (cromossomo) cresce e o número de threads aumenta.

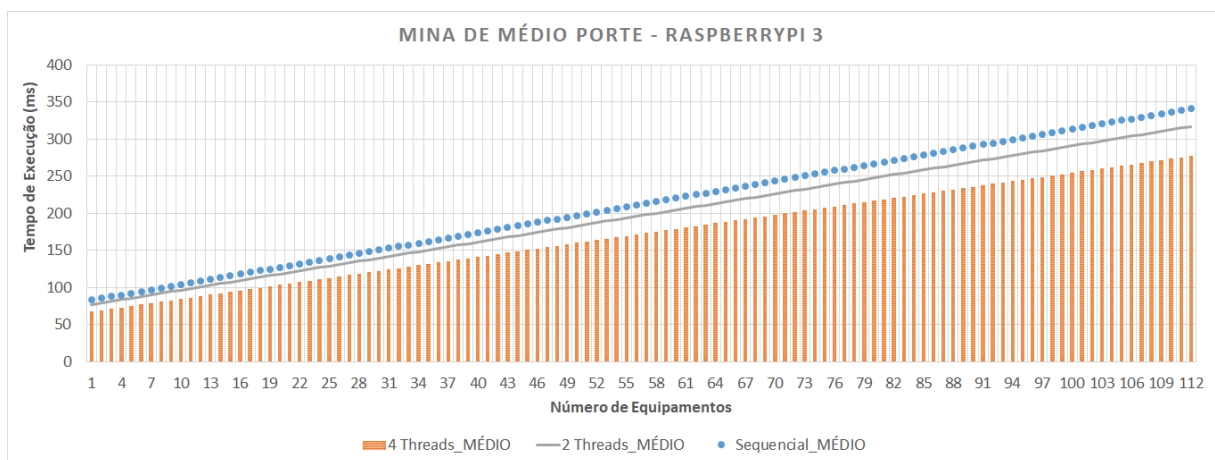


Figura 31 – Mina de Médio Porte - Raspberry Pi3
 Fonte: Acervo do Autor

Os resultados mostrados na Figura 32 para uma mina com configuração de grande porte demonstram que a função de migração do algoritmo genético com ilhas para um número maior de threads executadas e pelo tamanho do indivíduo (cromossomo) já ocasiona perda de desempenho. O tempo de execução para um determinado número de equipamentos apresenta melhor desempenho para o experimento com 2 threads do que sequencial do que o de 4 threads comparado com o de 2 threads o experimento de 4 threads.

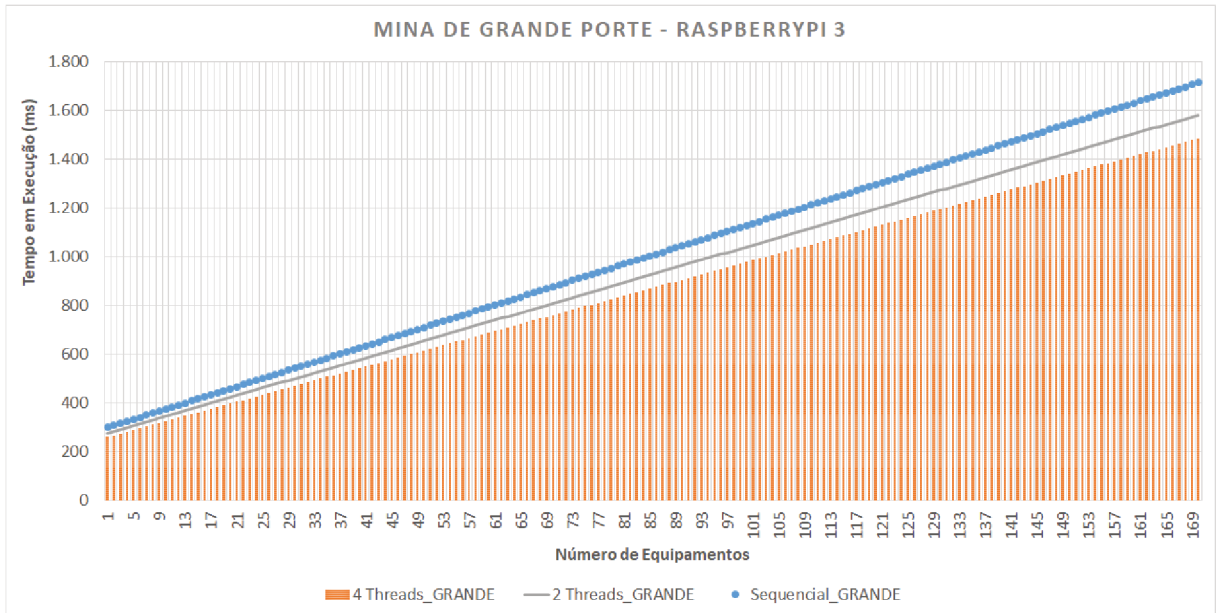


Figura 32 Mina de Grande Porte - Rasperry Pi3
 Fonte: Acervo do Autor)

Os resultados mostram que o paralelismo do algoritmo genético em ilha proposto reduz o tempo de execução, dependendo do número de equipamentos e da complexidade da mina. Melhores resultados foram obtidos para minas de médio e grande porte quanto o número de equipamentos tende para o valor máximo.

Hardware / Porte da Mina	Mina de Pequeno Porte				Mina de Médio Porte				(Max. 170 Equipamentos)			
Raspberrypi 2	70.620	69.420	69.077	NA	560.600	528.776	471.851	a	2.091.884	1.935.738	1.764.320	NA
Raspberrypi 3	44.703	44.229	41.264	NA	341.289	316.883	277.230	NA	1.714.269	1.581.239	1.485.555	NA
BananaPI M3	47.147	45.429	43.774	43.696	285.797	262.521	242.597	219.901	1.424.012	1.374.679	1.252.945	1.104.403
Nº de Threads	Sequencial	2	4	8	sequencial	2	4	8	Sequencial	2	4	8

Figura 33 Melhor resultado de desempenho
 Fonte: Acervo do Autor)

A Figura 34 exibe os resultados da mina de pequeno porte onde o desempenho da Raspberrypi 2 teve pior desempenho se comparado com a Banana Pi e Raspberrypi 3. Esse resultado apresenta o tamanho do indivíduo (cromossomo) versus a execução do algoritmo por um determinado número de threads (uma thread em cada núcleo). Nesse experimento, não houve interferência de perda de desempenho ocasionado pela função de migração do algoritmo genético paralelo no modelo de ilhas. Não foram executados os testes com 8 Threads na Raspberrypi 2 e 3 sendo que as mesmas não possuem esse recurso.

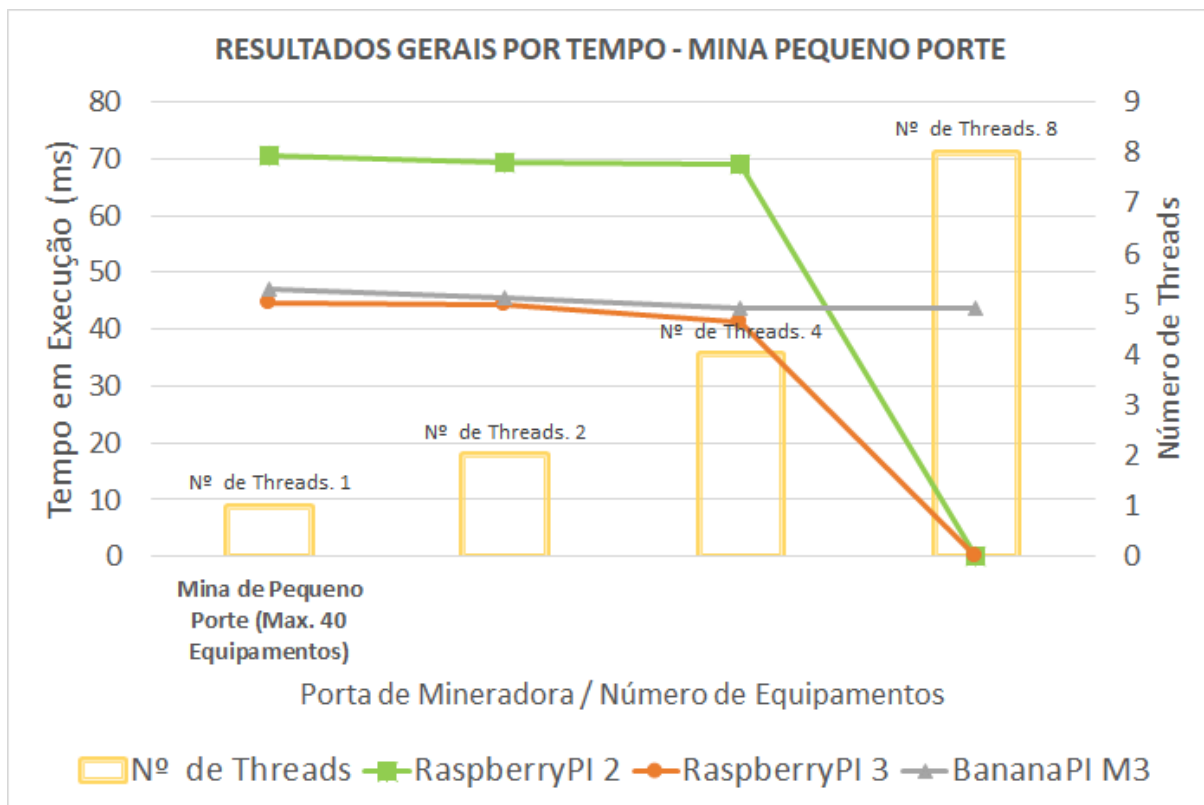


Figura 34 – Gráfico de resultado geral por tempo - mina de pequeno porte
 Fonte: Acervo do Autor)

A Figura 35 demonstra os resultados obtidos para uma mina de médio porte de até 110 equipamentos. O melhor resultado obtido foi com a execução no Banana Pi para as opções: sequencial, 2, 4 e 8 threads. Nesse experimento, não ocorreu perda do desempenho devido ao tamanho do indivíduo (cromossomo).

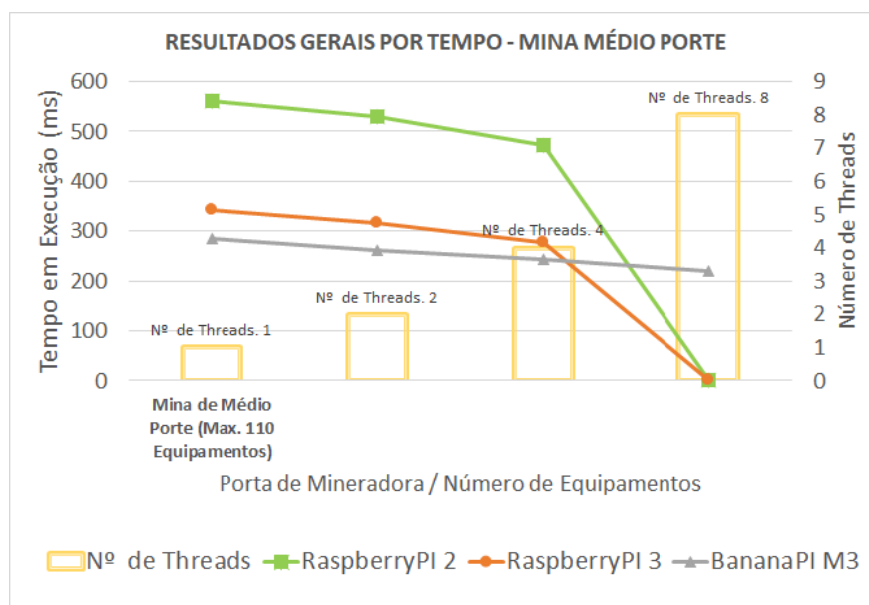


Figura 35 – Gráfico de resultado geral por tempo - mina de médio porte
 Fonte: Acervo do Autor)

A Figura 36 mostra o resultado da execução do algoritmo genético paralelo para uma mina de grande porte. O melhor resultado foi obtido também pela maior quantidade de threads executadas, com a BananaPI. Independente do número de threads, nenhum dos experimentos conseguiu executar o algoritmo em um tempo inferior a um segundo (1s), que era o tempo máximo permitido de modo que o motorista tenha tempo suficiente para tomar a decisão de mudar sua rota, caso seja necessário.

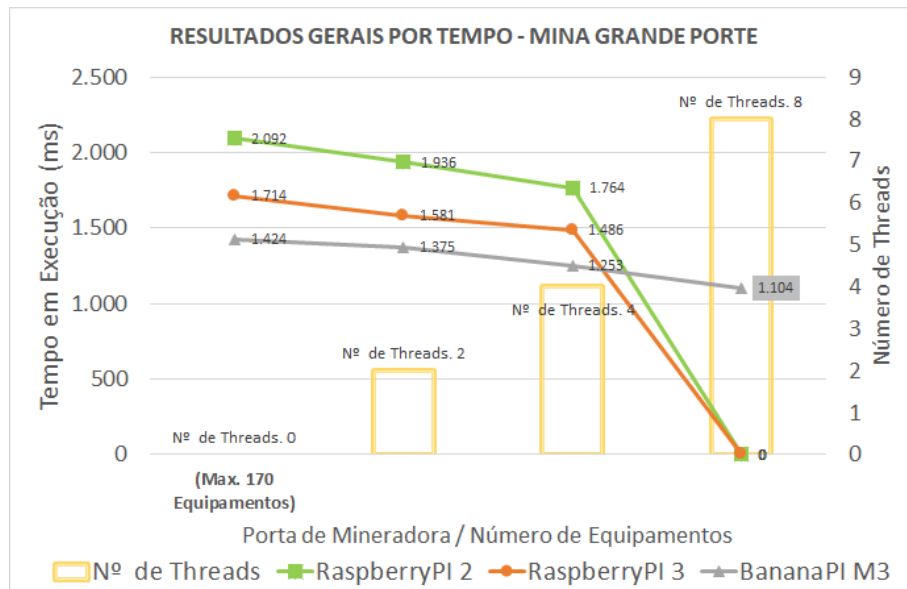


Figura 36 – Gráfico de resultado geral por tempo - mina de grande porte
 Fonte: Acervo do Autor)

Discussão e Conclusões

De acordo com os resultados dos testes, a utilização de algoritmos genéticos para a solução de problemas de grau NP-Complexo, em equipamentos embarcados, é viável.

Conforme o número de equipamentos aumenta, cresce o tempo de execução do algoritmo genético. No entanto, observa-se que o desempenho melhora conforme o indivíduo (cromossomo) cresce e o número de threads aumenta, isto é, a medida que o cromossomo cresce executa-se instâncias com maiores cargas computacionais, conforme constatado por Rosa (2015).

Os resultados obtidos com os sistemas embarcados mostraram que é possível reduzir os custos com infraestrutura robusta de servidores e aquisição de licenciamento de sistema de otimização de despacho na mineração. Isso representa milhares de reais de economia para minas de pequeno porte e redução de custos para minas de médio e grande porte.

Conforme os resultados demonstrados, a paralelização de algoritmos genéticos em hardwares embarcados traz novas possibilidades de aplicação de inteligências computacionais para hardwares de baixo custo, permitindo o processamento descentralizado na otimização de despacho em mineração a céu aberto com múltiplas rotas.

8.1 Sugestões de Trabalhos Futuros

Como sugestões de trabalhos futuros podem ser feitas as seguintes melhorias:

- ❑ Ganho de desempenho poderá ser obtido retirando-se a função de migração, conforme proposto por Whitley, Rana e Heckendorn (1998) aplicou em um modelo VRP;
- ❑ Buscar meios para melhorar o desempenho do algoritmo genético proposto eliminando possíveis overheads de software;
- ❑ Customização de um sistema operacional, isto é, compilar o kernel e os módulos indispensáveis para a paralelização do algoritmo proposto;

- ❑ Utilizar um sistema de tempo real;
- ❑ Otimizar o uso do OpenMP.

Referências

- ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1989. ISBN 0-8053-0177-1.
- ALVARENGA, G. B. **Despacho Ótimo de Caminhões Numa Mineração de Ferro Utilizando Algoritmo Genético com Procesamento Paralelo**. Dissertação (Mestrado), 1997.
- ANDERSON, T. E. et al. Scheduler activations: Effective kernel support for the user-level management of parallelism. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 25, n. 5, p. 95–109, set. 1991. ISSN 0163-5980. Disponível em: <<http://doi.acm.org/10.1145/121133.121151>>.
- ASANOVI, K. et al. **The Landscape of Parallel Computing Research: A View from Berkeley**. [S.l.], 2006.
- BENGTSSON RASTISLAV GALIA, T. G. C. H. L.; KOHL, N. Railway crew pairing optimization. **Springer-Verlag Berlin Heidelberg**, 2007.
- CHENG, J.; GROSSMAN, M.; MCKERCHER, T. **Professional CUDA C Programming**. Wiley, 2014. (EBL-Schweitzer). ISBN 9781118739327. Disponível em: <https://books.google.com.br/books?id=_Z7rnAEACAAJ>.
- COMPUTERWORLD. **8 tendências que irão impactar o mercado de tecnologia em 2018**. Acessado em 15/12/2017. Disponível em: <<http://computerworld.com.br/8-tendencias-que-irao-impactar-o-mercado-de-tecnologia-em-2018>>.
- LINDEN, R. **Algoritmos Genéticos (2a edição)**. BRASPORT, 2012. ISBN 9788574523736. Disponível em: <<https://books.google.com.br/books?id=it0kv6UsEMEC>>.
- MACHADO, F.; MAIA, L. **Arquitetura de sistemas operacionais**. LTC, 1996. ISBN 9788521610977. Disponível em: <<https://books.google.com.br/books?id=aAE9AAAACAAJ>>.
- MOORE, G. E. Cramming more components onto integrated circuits. **Electronics, Volume 38, Number 8**, 1965.
- OCHI, L. S. et al. A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. **Future Generation Computer Systems**, v. 14, n. 5, p. 285 – 292,

1998. ISSN 0167-739X. Bio-inspired solutions to parallel processing problems. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X9800034X>>.

PACHECO, P. S. **Parallel Programming with MPI**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996. ISBN 1-55860-339-5.

PAZ, E. C. A survey of parallel genetic algorithms. **Calculateurs Paralleles, Reseaux et Systems Repartis**, v. 10, n. 2, p. 141–171, 1998.

QUINN, M. J. **Parallel Programming in C with MPI and OpenMP**. [S.l.]: McGraw-Hill Education Group, 2003. ISBN 0071232656.

ROSA, M. C. C. Mateus Fontoura Gomes da. Estudo preliminar sobre a escalabilidade de um algoritmo genético paralelizado com openmp. 2015.

ROSÁRIO, D. A. N. do. **Escalabilidade paralela de um algoritmo de migração reversa no tempo (rtm) pré-empilhamento**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2012.

SNIR, M. et al. **MPI-The Complete Reference, Volume 1: The MPI Core**. 2nd. (revised). ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262692155.

SUTTER, H.; LARUS, J. Software and the concurrency revolution. v. 3, n. 7, p. 54–62, set. 2005. ISSN 1542-7730 (print), 1542-7749 (electronic).

TANENBAUM, A. **Sistemas operacionais modernos**. Prentice-Hall do Brasil, 2010. ISBN 9788576052371. Disponível em: <<https://books.google.com.br/books?id=nDatQwAACAAJ>>.

TORRES, M. F. D. **Rede de Sistemas Embarcados que utilizam Algoritmos Genéticos aplicado na otimização de Despacho em Mineração com Múltiplas Rotas**. Dissertação (Mestrado) — UFU - Universidade Federal de Uberlândia, 2017.

TORRES, M. F. D.; NETO, G. D. Rede de sistemas embarcados que utilizam algoritmos genéticos aplicado na otimização de despacho em mineração com múltiplas rotas. **CONEM - Contresso Nacional de Engenharia Mecânica**, 2016.

WHITLEY, D.; RANA, S.; HECKENDORN, R. B. The island model genetic algorithm: On separability, population size and convergence. **Journal of Computing and Information Technology**, v. 7, p. 33–47, 1998.

YAGHMOUR, K. et al. **CONSTRUINDO SISTEMAS LINUX EMBARCADOS**. ALTA BOOKS. ISBN 9788576083436. Disponível em: <<https://books.google.com.br/books?id=nTLHPwAACAAJ>>.