INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Minas Gerais
Campus Bambuí

COMPARAÇÃO DO DESEMPENHO SEQUENCIAL E PARALELO ENTRE DIFERENTES ARQUITETURAS COMPUTACIONAIS: RASPBERRY PI E UM COMPUTADOR DESKTOP

Paulinelly de Sousa Oliveira

Bambuí - MG
2017

Paulinelly de Sousa Oliveira

COMPARAÇÃO DO DESEMPENHO SEQUENCIAL E PARALELO ENTRE DIFERENTES ARQUITETURAS COMPUTACIONAIS: RASPBERRY PI E UM COMPUTADOR DESKTOP

Monografia apresentada ao Curso de Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *campus* Bambuí - Campus Bambuí, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Área de concentração: Computação de Alto Desempenho

Orientador: Laerte Mateus Rodrigues
Coorientador: Carlos Renato Nolli

Bambuí - MG
2017

Ribeiro, Marcos Roberto.
Classe LaTeX para Trabalhos Acadêmicos de Institutos Federais/ Marcos Roberto Ribeiro. 2017.
55 p. :il.
Orientador: Nome do Orientador.
Co-orientadora: Nome do Co-orientadora.
Monografia de Trabalho de Conclusão de Curso - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - Campus Bambuí, Curso Engenharia de Computação, 2017

1. Trabalho de conclusão de curso. 2. Latex. 3. Monografia. I. Ribeiro, Marcos Roberto. II. Título.

Paulinelly de Sousa Oliveira

COMPARAÇÃO DO DESEMPENHO SEQUENCIAL E PARALELO ENTRE DIFERENTES ARQUITETURAS COMPUTACIONAIS: RASPBERRY PI E UM COMPUTADOR DESKTOP

Monografia apresentada ao Curso de Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *campus* Bambuí - Campus Bambuí, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Bambuí - MG, 15 de Dezembro de 2017

Laerte Mateus Rodrigues
Orientador
Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *campus* Bambuí

Carlos Renato Nolli
Coorientador
Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *campus* Bambuí

Ciniro Aparecido Leite Nametala
Instituto Federal de Minas Gerais - *campus* Bambuí

Francisco Heider Willy dos Santos
Instituto Federal de Minas Gerais - *campus* Bambuí

A Deus, pela força para lutar e chegar até aqui.
Aos meus pais, à minha irmã e meus amigos, pelo incentivo e motivação para alcançar meus sonhos.
A todos que contribuíram e/ou torceram por mim.

Agradecimentos

Agradeço a Deus por concedido esta grande oportunidade e por ter me dado forças em todos os momentos difíceis. Agradeço aos meus pais (José Vicente e Vera Lúcia) e a minha irmã (Patrícia) por por terem acreditado em mim, por todo o apoio e confiança e por terem abraçado este sonho comigo. Meu orientador (Laerte), co-orientador (Renato Nolli) e os demais professores pela ajuda, paciência, conselhos e todo conhecimento que me passaram permitindo que eu chegasse aqui. Aos meus amigos, colegas de sala e de curso que me ajudaram direta ou indiretamente. A todos que de alguma forma contribuíram: meu MUITO OBRIGADO!!!

"A imaginação é mais importante que a ciência, porque a ciência é limitada, ao passo que a imaginação abrange o mundo inteiro."

(Albert Einstein)

"Por mais que a ciência evolua e que a tecnologia avance jamais ela vai decifrar a mente humana, pois cada cabeça é um mundo e cada ser humano uma história, jamais caberá numa tese ou num fundamento. Isso faz da humanidade e seu imaginário imensamente complexos e hierárquicos."

(Afonso Allan)

Resumo

Os sistemas embarcados estão cada vez mais presentes em nosso cotidiano, modelos (de propósito geral) como a Raspberry, Orange Pi, Beaglebone, entre outros, vem se tornando mais populares principalmente em aplicações domésticas e/ou de baixo custo. Este trabalho apresenta um comparativo de desempenho entre um computador *desktop* e uma placa Raspberry Pi 3B. Um algoritmo genético para resolução do problema do Caixeiro Viajante foi utilizado para a análise, sendo que foram testadas entradas de 5, 10, 15 até 100 cidades tanto de forma sequencial quanto de forma paralela na placa e no computador, a fim de verificar o desempenho de cada arquitetura de acordo com o tamanho da entrada. Uma amostra, de 50 execuções, com entrada fixa de 30 cidades também foi realizada em cada uma das arquiteturas (sequencial e paralelo) a fim de verificar aspectos como comportamento ao longo do tempo, entre outros. Os resultados mostram que o desempenho da Raspberry Pi é inferior ao do computador, resultado esperado, visto que se trata de uma arquitetura embarcada que possui recursos limitados, além de recursos como memória primária e secundária mais lenta que a de um computador, mas a placa apresentou melhor eficiência na utilização de seus núcleos em relação ao computador. Contudo a Raspberry Pi é uma alternativa barate e eficiente para sistemas de baixo custo ou que não tenham tempo de respota como fator crítico.

Palavras-chave: Raspberry Pi. Desempenho. Paralelo. Sequencial.

Abstract

Embedded systems are becoming more and more present in our daily lives, and general-purpose models such as Raspberry, Orange Pi, Beaglebone, among others, have become more popular in home and / or low-cost applications. This work presents a comparative performance between a desktop computer and a Raspberry Pi 3B card. A genetic algorithm for solving the traveling salesman problem was used for the analysis, and 5, 10, 15, and 100 cities were tested both sequentially and in parallel on the board and computer to verify performance of each architecture according to the size of the input. A sample of 50 executions with fixed input of 30 cities was also performed in each of the architectures (sequential and parallel) in order to verify aspects such as behavior over time, among others. The results show that the performance of Raspberry Pi is lower than that of the computer, an expected result, since it is an embedded architecture with limited resources, as well as features such as primary and secondary memory slower than a computer, presented better efficiency in the use of their cores in relation to the computer. However, Raspberry Pi is a cheap and efficient alternative for low-cost or non-responsive systems as a critical factor.

Keywords: Raspberry Pi. Performance. Parallel. Sequential.

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABNT - Associação Brasileira de Normas Técnicas

IFMG - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais

SQL - *Structured Query Language*

TCC - Trabalho de conclusão de curso

Lista de Algoritmos

Lista de Códigos

Sumário

1 Introdução

Atualmente está sendo cada vez mais necessário o aumento da capacidade de processamento dos computadores, devido a novas necessidades que vêm surgindo nas mais diversas atividades e áreas do conhecimento - principalmente com simulações de problemas relacionados à ciência e à engenharia. Nesse contexto o conjunto de dados a ser tratado torna-se cada vez maior devido à facilidade de armazenamento destes dados, o que gera uma grande demanda de recursos computacionais para armazenamento, recuperação, processamento e transmissão de informações. Consequentemente a extração de informações relevantes torna-se ainda mais trabalhosa. (LIMA, 2016, p. 16).

Ao longo dos anos várias técnicas foram utilizadas para melhorar a organização do computador a fim de aumentar a velocidade de processamento como o uso de memória virtual, unidades de armazenamento externas que não usam discos, hierarquia de barramentos, redução do tamanho dos componentes do microprocessador (o que reduz a distância entre os componentes), *pipelines*, etc. Na arquitetura também houve melhorias, como conjunto de instruções RISC, previsão de desvio, análise do fluxo de dados, execução especulativa, entre outros. Ainda assim o equilíbrio do desempenho foi necessário para evitar que as melhorias de um componente fossem anuladas pelos atrasos de outros, como exemplo, pode-se citar o desenvolvimento hierarquia de memória (que consiste em múltiplos níveis de memória com diferentes velocidades e tamanhos), memória cache, caminhos de dados entre memória e processador mais largos e chips de memória mais inteligentes, isso porque o desenvolvimento da memória principal não acompanhou o do processador. Contudo, mesmo com estes recursos ainda existem limites físicos como a frequência suportada pelo silício. (STALLINGS, 2002). Desta forma, para contornar o problema os projetistas de processadores começaram a colocar vários núcleos num mesmo chip a fim de ser possível o processamento paralelo real.

> Para resolver esses problemas, os designers de processadores passaram a projetar circuitos que pudessem processar dados em paralelo, ou seja, ao invés de tentarem aumentar o poder de processamento de um único núcleo do processador, incluindo mais transistores, os designers passaram a considerar a criação de processadores com muitos núcleos em um único circuito integrado. Tais circuitos são conhecidos como processadores *multicore*. Ao tirar proveito do paralelismo com processadores compostos por vários núcleos, foi possível elevar o poder de processamento a taxas mais altas do que as que vinham sendo alcançadas quando apenas um núcleo era utilizado. (LIMA, 2016).

Desde os trabalhos de Neumann (1945) *apud* Navaux, Rose e Pilla (2011) arquiteturas paralelas já eram consideradas soluções com maior capacidade de processamento em execuções com grandes volumes de processamento. Nos anos 70, quando as tecnologias computacionais passaram a não ser mais suficientes para suprir as demandas de

processamento, deu-se início da utilização de técnicas de concorrência para se ter maior desempenho. Limitações como altos custos, tendenciaram o desenvolvimento de sistemas com vários núcleos juntos, nos chips de microprocessadores, processadores e até mesmo dos supercomputadores. Assim tem-se várias unidades ativas (ou núcleos) trabalhando de forma paralela na mesma aplicação para que se tenha uma redução no tempo de execução da mesma. (NAVAUX; ROSE; PILLA, 2011).

Mesmo com avanços consideráveis na arquitetura e na organização de computadores, a nível de *software* também foram desenvolvidas técnicas para se obter mais facilmente melhores resultados na resolução de problemas, isso porque existem alguns problemas que não podem ser solucionados com melhorias no *hardware*. Essas técnicas nem sempre garantem o melhor resultado, mas sim garantem bons resultados. Os algoritmos bio-inspirados (que são uma sub-classe da Computação Natural) tem grande importância nesse cenário, por serem utilizados em vários tipos de problemas, como reconhecimento de padrões e otimização, por exemplo. A Figura 1 mostra divisões e subdivisões da computação natural. A Figura 2 mostra os principais algoritmos dentro da sub-classe algoritmos bio-inspirados. (ROSSI, 2009).

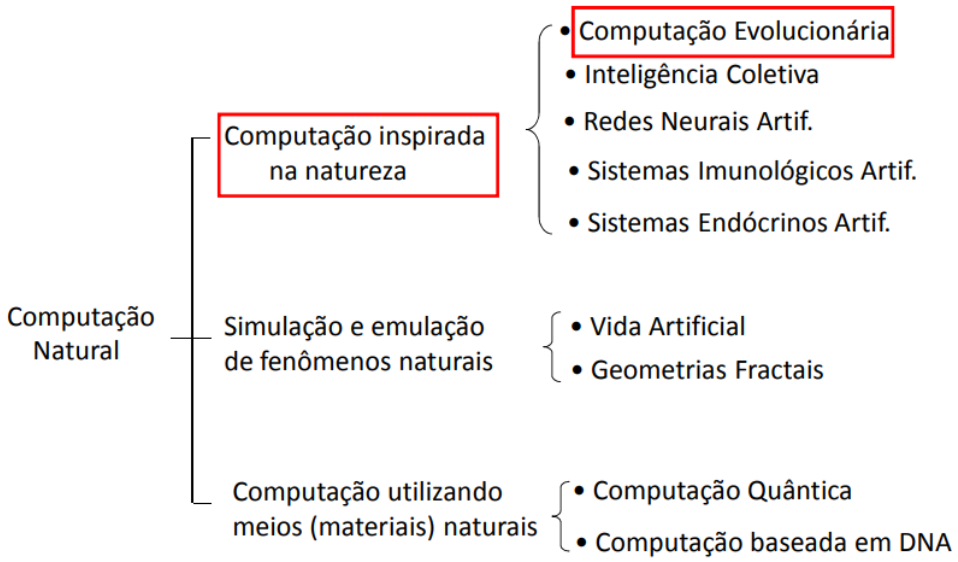

Figura 1 – Computação Natural.
Possui três subdivisões, sendo elas computação inspirada na natureza (ou algoritmos bio-inspirados: computação evolucionária, inteligencia coletiva, redes neurais artificiais, sistemas imunológicos artificiais e sistemas endócrinos artificiais), simulação e emulação de fenômenos naturais (vida artificial e geometriais fractais) e a computação utilizando meios físicos naturais (computação quântica e computação baseada em DNA). Fonte: (PAPPA, 2015).

Além destes avanços, Fuller e Millett (2011) mostram que em função do ciclo virtuoso em que a tecnologia se encontra a aplicação da computação em diversos outros sistemas podem ser vistos, como por exemplo, nos atuais *smartphones*, smarTV's e sistemas embarcados que possuem componentes para computação de propósito geral e específico.

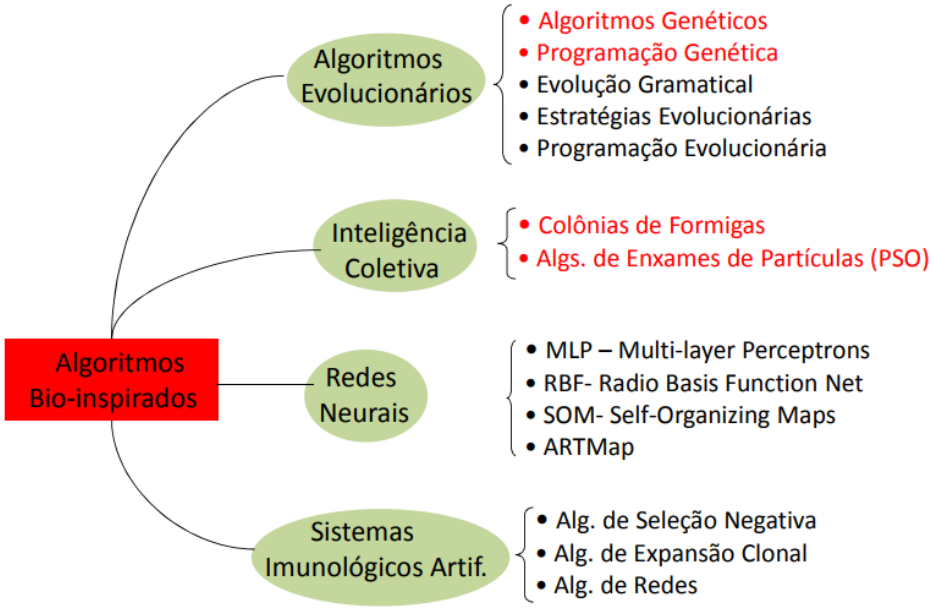

Figura 2 – Algoritmos bio-inspirados.
São divididos em: algoritmos evolucionários, inteligência coletiva, redes neurais e sistemas imunológicos artificiais. Fonte: (PAPPA, 2015).

Um destes modelos é o Raspberry Pi (que, basicamente é um pequeno computador) que possui processador, memória, comunicação com dispositivos de entrada e saída, entre outros.

Sendo assim, este trabalho propõe o desenvolvimento de uma metodologia para comparar o desempenho de programas de forma sequencial e também de forma paralela em duas diferentes arquiteturas, ambas de propósito geral: plataforma Raspberry Pi e um computador *desktop*.

1.1 Objetivos

Para o desenvolvimento deste trabalho, foram definidos alguns objetivos (gerais e específicos) aos quais são apresentados nas subseções a seguir.

1.1.1 Objetivo Geral

Comparar o desempenho computacional paralelo e sequencial do Raspberry Pi em relação à computadores (*desktops*).

1.1.2 Objetivos Específicos

a. Definir o algoritmo a ser utilizado no trabalho a fim de utilizar o máximo possível do *hardware* em ambos os cenários;

b. Adaptar ou implementar o algoritmo para execução utilizando programação paralela.

c. Avaliar o desempenho computacional nas 2 arquiteturas propostas.

1.2 Justificativa

Este trabalho tem como justificativa o uso de um computador de baixo custo que apresenta desempenho significativo e que está se tornando cada vez mais popular nos dias atuais e, sua comparação com um computador que já é utilizado. Há também a utilização de uma técnica que foi desenvolvida como solução para melhorar o desempenho dos processadores, que é a programação paralela, aproveitando todos os núcleos disponíveis. Como foi apresentado anteriormente, processadores multicore são uma das soluções desenvolvidas para melhorar a capacidade de processamento visto que já não está sendo possível diminuir o tamanho físico dos transistores com a atual tecnologia de semicondutores. É importante ressaltar que na nossa instituição não há trabalhos científicos utilizando o Raspberry Pi, sendo assim este trabalho serve de alicerce para futuros trabalhos.

1.3 Organização do Trabalho

Para um melhor entendimento deste trabalho, o mesmo foi dividido em alguns capítulos. O capítulo 2 apresenta o referencial teórico, onde são descritos todos os termos e elementos necessários para uma melhor compreensão do trabalho. O capítulo 3 apresenta a revisão de literatura, em que é mostrado os trabalhos existentes correlatos a este, quais foram suas metodologias e resultados e, ao final como este trabalho se difere dos demais. O capítulo 4 apresenta os materiais e métodos utilizados para o desenvolvimento deste trabalho, em outras palavras são descritas as técnicas e métricas utilizadas neste trabalho. O capítulo 5 apresenta os resultados e discussões obtidos após a execução da metodologia descrita no capítulo 4 e faz uma comparação com os resultados dos trabalhos descritos no capítulo 3. O capítulo 6 apresenta as conclusões obtidas com base nos resultados descritos no capítulo 5 e nos objetivos proposto no capítulo 1, quais limitações foram notadas, quais possíveis trabalhos futuros podem desenvolvidos. Por fim, o capítulo de referenciais apresenta a identificação e origem dos trabalhos consultados para o desenvolvimento deste.

2 Referencial Teórico

Para o desenvolvimento deste trabalho, faz-se necessário o entendimento de alguns conceitos, que forão abordados de maneira direta ou indireta no decorrer deste trabalho. Estes conceitos são apresentados nas subsseções seguintes.

2.1 Arquitetura RISC

A arquitetura RISC (*Reduced Instruction Set Computer*) surgiu a partir da necessidade de se ter todos os componentes de um processador no mesmo chip - com a até então CISC (*Complex Instruction Set Computer*) não era possível. Mas para isso algumas funcionalidades foram retiradas de dentro do processador, passando então a complexidade para o *software*, enquanto que as tarefas mais frequentes foram otimizadas e executadas diretamente pelo *hardware*, pois percebeu-se que a maior parte do trabalho era realizado por esse conjunto menor de instruções. Por exemplo, o suporte às linguagens de alto nível foram retiradas dos circuitos integrados e passados para o *software*, o suporte ao microcódigo também foi retirado e as instruções são executadas em apenas uma microinstrução. Além da quantidade de instruções que foram reduzidas, o tamanho das mesmas foram encurtados e sempre que possível deveria demorar apenas um ciclo de relógio para serem executadas. O microcódigo deu lugar a pequenas (e em formato fixo) instruções em *Assemly*, retirando assim o espaço da memória que era do microcódigo. Dessa forma, os compiladores passam a ter papel essencial para o bom funcionamento da arquitetura, pois estes tem toda a responsabilidade de otimizar os códigos durante o processo de compilação. (SILVA; ANTUNES, 2008).

Com a redução do tamanho das instruções foi possível também à implementação do *pipelining* (que não é viável em ambientes com instruções de complexidade variável). Este recurso permite a execução de várias instruções paralelamente, o que reduz a quantidade média de ciclos por instrução (CPI ou *Cycles Per Instruction*) e consequentemente o tempo de processamento dos programas (aumento de desempenho de processamento). Segundo Silva e Antunes (2008) outras duas características, que permitiram a redução de ciclos/instrução aumentando minimamente o tamanho do código na arquitetura RISC, são "a eliminação dos modos de endereçamento complexos e o aumento do número de registradores internos do processador". O aumento desses registradores têm a função de evitar a busca de variáveis locais na memória, pois estas são mantidas conforme a necessidade, em bancos de registradores sempre que sub-rotinas são chamadas. Azevedo e Oliveira (2014) apresentam algumas características dessa arquitetura:

1. Registradores: 16-32;

2. Tipos de dados: normalmente dois (inteiro e ponto flutuante);
3. Instruções: *Load/Store*, registradores gerais, operações sobre tipos de dados (*datatypes*) nos registradores;
4. Formato das instruções: tamanho fixo, dois tipos principais: *Load/Store* ou R:=R op R;
5. Codificação 1 instrução = 1 operando ou 1 operação;
6. Objetivo do projeto: minimizar tempo de execução da instrução;
7. Implementação: processador e *software* com alto grau de ligação, processador e cache rápidos para instruções, instruções levam 1 ciclo de *clock*, *pipeline* simples;
8. *Caching*: essencial para as instruções;
9. Filosofia: mover todas as funções para o *software*;
10. Dispositivos: iPod, iPhone, iPad, Palm e PocketPC, Nintendo, Gameboy, Sun SPARC.

2.2 Arquitetura ARMv7 e ARMv8-A

O ARMv7 possui possui arquitetura do tipo RISC de 32 *bits*, *pipeline* em três estágios e registradores *load/store*. Isso faz com que o processador nunca fique ocioso, pois sempre estará executando diferentes estágios de instruções: buscando, decodificando e executando. Essa é uma maneira simples e eficiente de se evitar conflitos entre estágios de outras arquiteturas. Outra característica é que este possui sete modos diferentes de execução (que são usados para tratamento de exceções, de interrupções, entre outras funcionalidades). (CRUZJúNIOR, 2013, p. 16). O Quadro 1 apresenta esses modos e suas descrições.

Item	Modo	Descrição
1	Usuário	Execução normal de programas.
2	Sistema	Executa rotinas privilegiadas do sistema operacional.
3	Supervisor	Modo protegido para o sistema operacional.
4	IRQ	Tratamento de interrupções comuns.
5	FIQ	Tratamento de interrupções rápidas.
6	Abort	Usado para implementar memória virtual ou proteção de memória.
7	Indefinido	Suporta emulação em *software* de co-processadores.

Quadro 1 – Sete modos de execução do ARMv7: usuário, sistemas, supervisor, IRQ, FIQ, *abort* e indefinido.

Fonte: (CRUZJúNIOR, 2013).

Todos estes modos apresentados na Tabela 01 também operam sob função especial (que pode ser ativada em tempo de execução) chamada de Thumb onde o ARMv7 passa a ser um core de 16 *bits* (o que aumenta o espaço da memória do programa). Este processador possui dois conjuntos de instruções (16 e 32 *bits*) sendo o de 16 *bits* é mais simples. Porém todo o processamento é feito em 32 *bits*, inclusive o acesso aos registradores. O ARMv7 possui melhor suporte às tradicionais operações de controle e às operações com ponto flutuante (se comparado com as famílias anteriores) principalmente devido às novas demandas de processamento gráfico (necessárias em jogos, processamento gráfico e outros). Há também a tecnologia NEON que melhora o DSP (*Digital Signal Processor*), podendo chegar a 400% de ganho no processamento digital de sinais. (CRUZJúNIOR, 2013, p. 17).

A Figura 3 apresenta a arquitetura ARMv8-A que têm como base a versão ARMv7 fazendo com que este tenha todas as características apresentadas anteriormente.

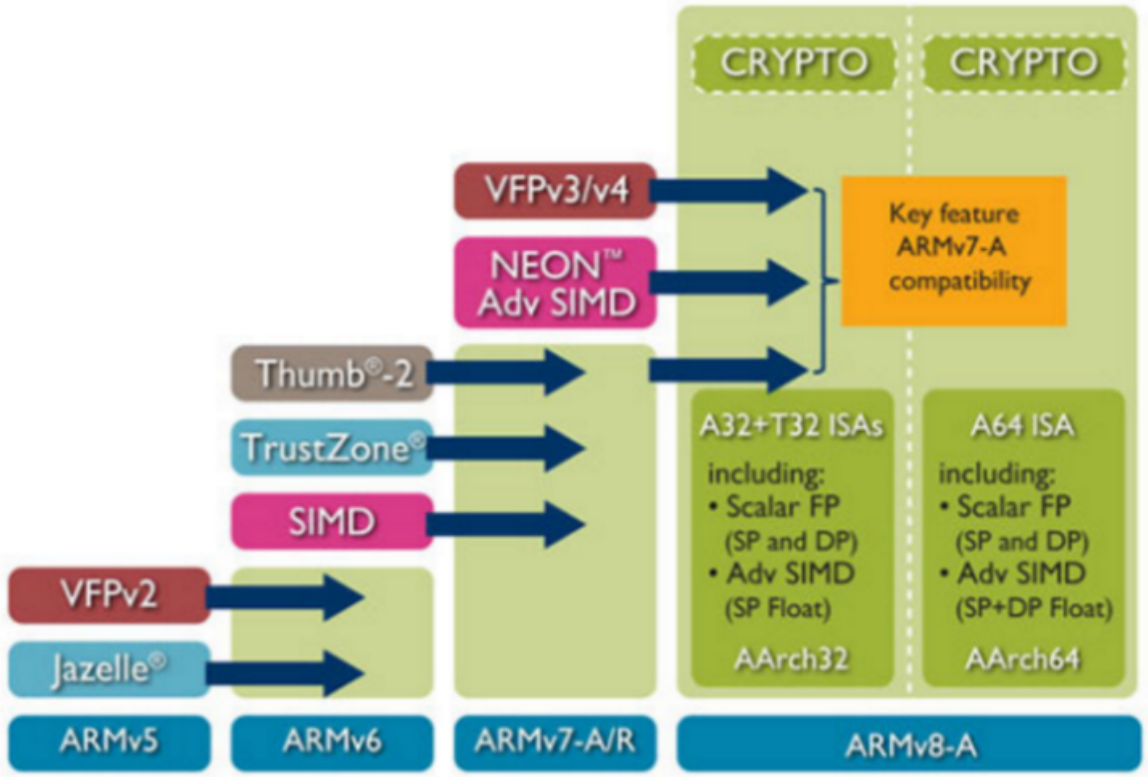

Figura 3 – Arquitetura ARM.
Diferenças e semelhanças entre as arquiteturas ARMv5, ARMv6, ARMv7 e ARMv8-A, que deixa claro a compatibilidade das versões mais novas com as anteriores. Fonte: (ARM-LTD., 2017).

O ARMv8-A tem sua arquitetura em 64 *bits* (sendo compatível com as versões 16 e 32 *bits* também) e a adição de instruções de 32 *bits* permite ainda a otimização para requisitos emergentes. Um maior intervalo de endereçamento e instruções permitem a computação de novas categorias de aplicações para *smartphones* e *tablets*. Outras características dessa arquitetura, descritas pela ARM-Ltd. (2017), que podem ser citadas são:

1. Registradores de uso geral de 64 *bits*, SP (*stack pointer*) e PC (*program counter*);

2. Processamento de dados de 64 *bits* e endereçamento virtual estendido;

3. Os estados de execução suportam três conjuntos de instruções principais:

    a) AArch32: um conjunto de instruções de comprimento fixo de 32 *bits*, aprimorado através das diferentes variantes de arquitetura;

    b) T32: (Thumb) introduzido como um conjunto de instruções de comprimento fixo de 16 *bits*, posteriormente aprimorado para um conjunto de instruções de 16 e 32 *bits* de comprimento misto na introdução da tecnologia Thumb-2;

    c) AArch64: é um conjunto de instruções de comprimento fixo de 64 *bits* que oferece funcionalidade semelhante aos conjuntos de instruções ARM e Thumb.

2.3 Arquitetura x86

Essa arquitetura teve seu nome (x86) originado do primeiro chip de 16 *bits* da Intel, o 8086, que evoluiu para os modelos 80186, 80286, 80386 e 80486 todos com o sufixo "86"indicando a compatibilidade do conjunto de instruções com as versões anteriores. Inicialmente esta arquitetura tinha seu conjunto de instruções do tipo CISC e devido as necessidades do mercado, passou a utilizar instruções híbridas RISC/CISC. (MARMITT, 2017). Motyczka et al. (2013) explica que dessa forma "eles recebem instruções CISC, que são convertidas em instruções RISC para serem trabalhadas internamente buscando aperfeiçoar o processamento através do uso de *pipeline*."São geradas micro operações parecidas com as instruções RISC, tendo-se assim uma máquina RISC dentro de uma x86. Mesmo sendo desenvolvido pela Intel, outras empresas passaram a utilizar a arquitetura x86 e, com o passar dos anos esta se tornou padrão para computadores, notebooks e supercomputadores, sempre mantendo compatibilidades com as versões anteriores da arquitetura. A família x86 define um conjunto de tipos de instruções: movimentação de dados, aritmética e lógica, sequenciamento, manipulação de *strings*, manipulação de *bits*, suporte a linguagens de alto nível, entre outras. (MOTYCZKA et al., 2013).

Mano (1993) *apud* Lorenzoni (2012) cita as principais características de processadores CISC:

1. Número elevado de instruções (entre 100 e 250 instruções);

2. Algumas instruções desempenham tarefas específicas e não são frequentemente utilizadas;

3. Grande variedade de modos de endereçamento;

4. Instruções com comprimento variável, assim utilizam diferentes números de *bits* para codificar as instruções que depende: quantidade de entradas da instrução, modos de endereçamento utilizados, entre outros fatores;

5. Instruções que manipulam operandos na memória.

2.4 Sistemas Embarcados

Os sistemas embarcados, também conhecidos como sistemas embutidos, são definidos como sistemas computacionais para uso específico ou dedicados. Estes estão se tornando cada vez mais populares e presentes na vida humana, principalmente por causa de algumas de suas vantagens: economia de energia, portabilidade, complexidade de processamento e baixo custo. A computação ubíqua - que diz respeito ao suporte computacional contínuo e permanente ao ser humano - tem como base os sistemas embarcados. Os núcleos de processamento de sistemas embarcados podem ser de propósito geral - que disponibilizam no *hardware* várias funções já implementadas, executam as mais variadas aplicações e funcionam com a execução de um programa - na qual tarefas como buscar, decodificar e executar instruções afetam diretamente o desempenho; ou podem ser de uso específico, possuindo circuitos integrados que são desenvolvidos para um conjunto específico de funções (ASIC - *Aplication Specific Integrated Circuit*). (CRUZJúNIOR, 2013).

Os projetos de sistemas embarcados podem apresentar uma grande flexibilidade não apenas do ponto de vista da programação, *software*, mas também em relação à parte física, *hardware*, especialmente quando implementados em sistemas reconfiguráveis. *Hardware* específico ou dedicado, normalmente é separado para tarefas que exigem alto poder de processamento, assim as demais funcionalidades são implementadas por meio de *software* ((WEI et al., 2008) *apud* (CRUZJúNIOR, 2013, p. 11)). Essa característica faz com que os sistemas de uso específico apresentem um desempenho melhor (na execução de tarefas) do que os sistemas de propósito geral.

2.5 Raspberry Pi

Basicamente é um pequeno computador - que possui memória primária e secundária, CPU (unidade central de processamento), GPU (unidade de processamento gráfico), sistema operacional (SO), conectores USB (*Universal Serial Bus*), saída de vídeo, saída de áudio, interface *Ethernet*, entre outros – de baixo custo e grande possibilidade de aplicação em problemas reais. Possui ainda pinos (GPIO – *General Purpose Input/Output*) que possibilitam o recebimento e o envio de sinais elétricos. (SHEIDEMANTEL, 2015, p. 3). Coutinho (2016, p. 19) afirma que “desenvolvida pela fundação Raspberry Pi, no Reino Unido, a plataforma Raspberry é um computador de baixo custo, pequeno, *Open Source* e

com interfaces para vários periféricos.” A FundaçãoRaspberryPi (2017) define esta placa como "um pequeno computador capaz de ser usado em projetos eletrônicos, e para muitas das coisas que seu PC de mesa faz, como planilhas, processamento de texto, navegar na internet e jogar jogos. Ele também reproduz vídeo de alta definição."Suas principais características são:

1. 1.2GHz 64-*bits quad-core* ARMv8-A CPU com memória *cache* de 32kB nível 1 e 512kB nível 2;
2. 1GB RAM LPDDR2 (900 MHz);
3. 4 portas USB 2.0;
4. 40 pinos GPIO;
5. Porta *Full* HDMI;
6. Porta 10/100 *Ethernet*;
7. *Combined* 3.5mm *audio jack and composite video*;
8. Interface de câmera (CSI);
9. *Display* interface (DSI);
10. Slot para cartão Micro SD (agora *push-pull*, em vez de *push-push*);
11. VideoCore IV (Núcleo gráfico 3D) - suporte a OpenGL ES 2.0, acelerador de *hardware* OpenVG, alta capacidade de codificação e decodificação em perfil 1080p30 H.264. A GPU tem capacidade de 1Gpixel/s, 1.5Gtexel/s ou 24GFLOPs em processamento de propósito geral e apresenta várias filtragens de texturas, além de também infra-estrutura DMA (*Direct Memory Access*);
12. Completa compatibilidade com os Raspberry Pi 1 e 2. A Figura 4 apresenta a distribuição dos componentes na placa.

Coutinho (2016, p. 20) afirma que “existem várias versões diferentes de sistemas operacionais (SOs) compatíveis com a plataforma, sendo que o SO do Raspberry é instalado no cartão SD do dispositivo.” Para fins de uso normal da placa, é recomendado a utilização da distribuição Linux Raspbian, recomendação esta feita pela fabricante que ainda acrescenta: “Raspbian é um sistema operacional livre baseado no Debian, otimizado para o hardware Raspberry Pi. Raspbian vem com mais de 35.000 pacotes: *software* pré-compilado empacotado em um formato agradável para fácil instalação em seu Raspberry Pi.” (FUNDAçãORASPBERRYPI, 2017). A Tabela 1 apresenta os dados de necessários para a alimentação da placa. Os pinos GPIO podem fornecer até 50mA com segurança (note que isso significa que 50mA distribuídos em todos os pinos: um pino individual GPIO só pode seguramente fornecer até 16mA).

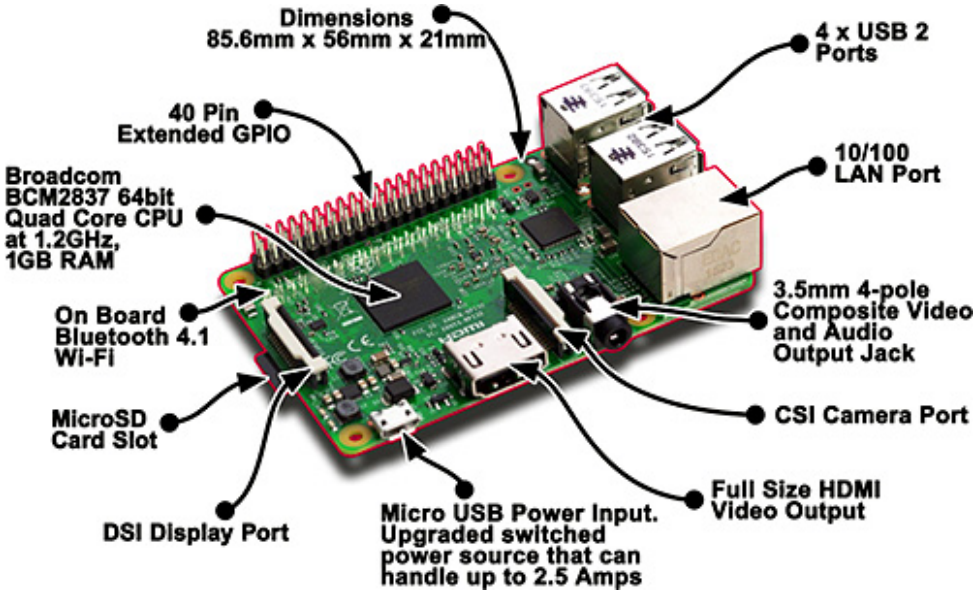

Figura 4 – Raspberry Pi 3 Modelo B.
Componentes da Raspberry Pi 3 modelo B sendo estes de entrada e saída (gpio, ethernet, USB, HDMI, interface CSI e DCI, conector audio, conector fonte de energia, slot SD card), bluetooh 4.1 e Wi-Fi. Fonte: (FUNDAçãORASPBERRYPI, 2017).

Tabela 1 – Modelos e características da raspberry, sendo capacidade recomendada de alimentação, corrente máxima consumida pelos periféricos USB e o consumo de corrente ativa típico na placa.

Raspberry Pi	**Capacidade recomendada da fonte de alimentação**	**Corrente máxima consumida pelos periféricos USB**	**Consumo de corrente ativa típico na placa**
Model A	700mA	500mA	200mA
Model B	1,2A	500mA	500mA
Model A+	700mA	500mA	180mA
Model B+	1,8A	600mA/1,2A (*switchble*)	330mA
2 Model B	1,8A	600mA/1,2A (*switchble*)	-
3 Model B	2,5A	1,2A	400mA
Zero W	1,2A	*Limited by PSU only*	150mA
Zero	1,2A	*Limited by PSU only*	100mA

Fonte: (FUNDAçãORASPBERRYPI, 2017).

2.6 Algoritmo Genético

Os Algoritmos Genéticos (AGs) são uma técnica de Inteligência Artificial baseado no princício da seleção e evolução natural de orgamimos biológicos (Darwinismo) em que os indivíduos que mais se adpatarem ao ambiente terão mais chance de sobreviver e se reproduzirem. Os AGs são amplamente aplicados em busca e otmização de resultados

(buscam a melhor solução para um determinado problema). ((PACHECO, 1999). (SILVA, 2001)).

Um AG trabalha de forma aleatória e orientada para algumas regras probabilísticas, de acordo com os mecanismos de reprodução e genética natural. Primeiro cria-se uma população inicial (em que cada indivíduo possui uma solução para o problema). Esta população é então avaliada para se conhecer quais são os melhores indivíduos (com melhor *fitness* ou melhor aptidão) e estes devem ter maiores chances de cruzamento para que se garanta a reprodução e propagação do material genético para as proximas gerações - uma alternativa é passar os melhores indivíduos direto para o proxima geração. Os indivíduos são implementados de forma que as características possam ser trocadas com outros indivíduos, tendo-se assim a reprodução. Cruzando-se os indivíduos, gera-se uma nova população que deve ser do mesmo tamanho da população inicial, a população antiga é substituida por esta nova. De forma percentual e aleatória aplica-se a mutação sobre a população (alteração dos cromossomos pode gerar indivíduos melhores). Esses procedimentos são repeditos até que se tenha uma solução aceitável ou até que se atinja um número N de gerações. A Figura 5 mostra um fluxograma básico e o Algoritmo 1 apresenta o pseudocódigo básico de um AG. ((HOLLAND, 1992) e (GOLDBERG, 1989) apud (ÁVILA, 2002)).

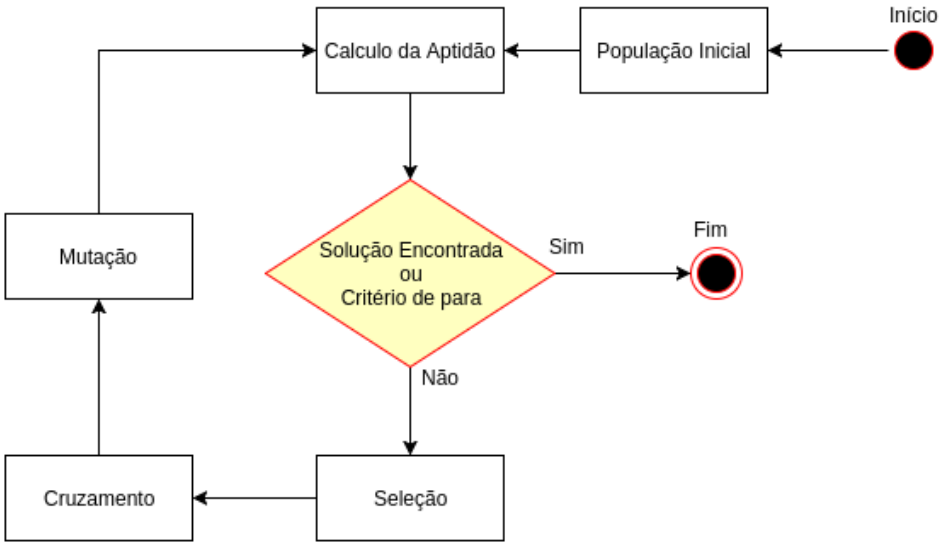

Figura 5 – Fluxograma de um AG.
Estrutura básica de um AG, que gera população inicial aleatória, avalia esta população, verifica a solução encontrada e o critério de para, faz a seleção dos melhores indivíduos, realiza o cruzamento (o que gera nova população), realiza a mutação e volta a verificar a solução e o critério de parada. Caso nenhum destes tenha sido atingido o algoritmo volta ao passo de seleção e assim continua até que se encontre um resultado ou atinja critêrio de pararda.

Os indivíduos devem possuir cromossomos, sendo que estes devem ter genes (que são caracterésticas ou informações sobre os indivíduos, em outras palavras é o material

Algoritmo 1 – Pseudocódigo do AG clássico

```
1: gerac ← 0; //inicializa contador gerações
2: Pop(gerac) ← Inicializa_populacao(); //população inicial gerada aleatoriamente
3: repita
4:     Avalia(Pop(gerac)); // avalia cada indivíduo da população atual
5:     (X1, X2) ← cruzamento(X1, X2); //cruzamento dos pais para gerar os filhos
6:     Xc ← mutacao(Xc); //operação de mutação dos filhos gerados
7:     gerac ← gerac + 1;
8:     Por(gerac) ← atualiza(Xm); //atualiza a população
9: até que g < criterio_parada
10: Print(Avalia(Pop(gerac)));
```

genético do indivíduo) que será trocado com outros indivíduos durante o cruzamento. Sendo assim, faz-se necessário uma função que seja capaz de avaliar o *fitness* dos indivíduos. A codificação do indivíduo e da função de avaliação tem grande importância, pois tem a responsabilidade de ligar o algoritmo com o problema real, avaliando quão boas são as características do indivíduo em relação ao problema. (NETO, 2011).

2.7 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV, do inglês: *Traveling Salesman Problem* - TSP) é um dos mais conhecidos problemas de otimização combinatória. Basicamente consiste no problema de encontrar a melhor rota possível (que pode ser menor caminho, caminho de menor custo, entre outros) em um conjunto de cidades, sendo que cada cidade só pode ser visitada uma única vez no trajeto. Quando o percurso entre duas cidades (ou nós) x e y não sofre interferência do sentido (Pxy = Pyx) o PCV é chamado de simétrico; quando há diferença em virtude do sentido do trajeto é chamdo de PCV assimétrico. O simétrico é em geral mais difícil de ser resolvido que o assimétrico. O PCV "tem sido usado como *benchmark* para avaliação de novos algoritmos e estratégias de solução que envolvem busca tabu, algoritmos genéticos, *simulated annealing*, redes neurais, entre outros."(CUNHA; BONASSER; ABRAHãO, 2002). O estudo e aplicação do PCV não se restringe no cenário de desempenho computacional, mas em inúmeras áreas (pesquisa operacional, matemática, física, biologia, inteligência artificial, entre outros) em vários tipos de problemas reais, como por exemplo, roteirização de veículos e *design* de circuitos.

> Sob a ótica de otimização, o PCV pertence à categoria conhecida como NP-difícil (do inglês NP-*hard*), o que significa que possui ordem de complexidade exponencial. Em outras palavras, o esforço computacional para a sua resolução cresce exponencialmente com o tamanho do problema (dado pelo número de pontos a serem atendidos). (CUNHA; BONASSER; ABRAHãO, 2002).

Problemas dessa categoria não podem ser resolvidos de maneira ótima em tempo viável, assim as técnicas atuais para resolução são conhecidas como heurísticas pois não

garantem um resultado ótimo, mas sim um resultado aproximado ou um bom resultado. Segundo Helsgaun (2000) citado por Cunha, Bonasser e Abrahão (2002), as heurísticas para resolução do PCV podem ser implementadas de duas formas:

1. Métodos de contrução de roteiros: os nós ou cidades vão sendo inseridos no roteiro de forma gradual e sequencial, sem que a solução parcial seja avaliada e/ou melhorada, assim a sequencia é montada e não é alterada mais;

2. Métodos de melhorias de roteiros: atravéz de um roteiro já obtido, com alguma técnica, busca-se melhorar o roteiro (diminuir o custo ou a distância do trajeto, por exemplo). Há autores que consideram uma terceira forma, que une a técnica de construção e a de melhoria de trajeto.

Segundo Laporte et al. (2000) e Reinelt (1994) *apud* por (CUNHA; BONASSER; ABRAHãO, 2002), a construção de roteiros pode ser dada através:

1. método do vizinho mais próximo;

2. método da inserção;

3. método das economias;

4. heurísticas baseadas em árvores de cobertura (*spanning trees*).

2.8 Sistemas Paralelos

Sistema paralelos são do tipo MIMD (*Multiple Instruction, Multiple Data*), pois possuem múltiplos fluxos de instruções e múltiplos fluxos de controle, como os processadores multicore, os computadores com múltiplos processadores e os clusters. O termo *multicore* é usado para definir sistemas com vários núcleos em um único chip e também sistemas com núcleos em chips separados ou independentes. Nos sistemas com mais de um núcleo que compartilham memória, o gerenciamento e sincronismo da mesma torna-se ainda mais rigoroso para evitar inconsistências nos dados. Para que os recursos implementados por sistemas paralelos sejam realmente utilizados, os *softwares* que são executados nos mesmos devem ter suporte para essas arquiteturas. Para isso, os algoritmos devem ter três características básicas: distribuição das tarefas para os diferentes processadores (mapeamento), execução compartilhada das tarefas conforme a dependência de dados (compartilhamento) e identificação dos dados que trafegam entre os processadores (identificação). (LIMA, 2016). Deve ser considerado que há diferentes níveis de paralelismo, dos quais pode-se destacar:

1. Paralelismo em nível de bit (bits de uma instrução são processados em paralelo);

2. Paralelismo em nível de instrução (*pipeline*);

3. Paralelismo em nível de dados (dados são acessador em paralelo e seus resultados são combinados);

4. Paralelismo em nível de tarefas (fluxos de controle independentes para cada tarefa).

3 Revisão de Literatura

Ramos, Ralha e Teodoro (2016) avaliam um cluster formado com 16 placas Raspberry Pi, totalizando 64 núcleos e 16GB de memória RAM – memória secundária de 16GB (cartão SD), SO Raspbian, interface de rede de 100Mb/s e todas as placas interligadas por um *switch ethernet* de 100Mb/s com 24 portas, para análise de imagens microscópicas. Essa avaliação compara processadores de baixo custo com CPUs multicores considerando tempo de execução, gasto energético, custo dos equipamentos e performance do cluster. Cada raspberry (modelo 2B) possui: processador Quad Core ARM7 Cortex 900MHz, 1GB RAM, interface de rede 100Mbits/s. Para instalação do SO foram utilizados cartões micro SD 16GB classe 10 (até 80MB/s). O cluster foi construído em quatro torres contendo quatro placas em cada, cada fonte de energia (de 5V DC e 2A) alimentou duas placas. Foi utilizada linguagem C++ e a paralelização foi feita utilizando a biblioteca MPI, a execução foi no modelo mestre/escravo. Foram utilizadas como entrada para os testes 512 imagens de 1K x 1K pixels, totalizando 6GB de dados de entrada.

Conclui-se que em consumo de energia, a placa (em capacidade máxima de processamento) obteve consumo menor que os computadores (mesmo em estado ocioso: apenas as aplicações do sistema), chegando a 2x menos energia. O cluster custou mais caro que os PCs, porem em desempenho este foi 2x mais rápido que a máquina com processador I7 e 10x mais rápido que a máquina com processador Core2Duo, tornando o cluster mais eficiente. Nas avaliações das máquinas em comparação com uma placa Raspberry Pi, esta apresentou desempenho pior que das máquinas (I7 e Code2Duo). Devido as memórias (primárias e secundárias) da placa serem mais lentas que as dos PCs, operações que realizam muitos acessos as estas memórias fazem com que o desempenho da Raspberry caia se comparado ao dos PCs (por exemplo: operações que visitam muitos pixels vizinhos do ponto analisado dentro de uma imagem). O processamento das imagens se deu pelas seguintes etapas: normalização, segmentação, extração de características, sumarização das características e clusterização.

Nunes et al. (2014) comparam o desempenho de sistemas reais e sistemas emulados em dispositivos com recursos limitados, nesse caso o Raspberry Pi. A análise de desempenho foi realizada em serviços web configurados na placa Raspberry Pi (utilizando RESTful e o *framework* CXF) – foram considerados os tempos de processamento de requisições, de empacotamento e desempacotamento de mensagens. Para avaliar o desempenho dos serviços web, o seguinte cenário foi proposto: a aplicação cliente gera uma sequência aleatória de números, envia para o servidor, que então recebe a mensagem ordena estes números e então envia de volta ao cliente os mesmos, ordenados e após recebidos, a aplicação cliente finaliza a conexão com o servidor. Foi utilizado um Raspberry Pi modelo B com cartão SD de 16GB e SO Linux Raspbian. Para o sistema emulado, foi utilizado o

software Qemu, que simulou um *hardware* equivalente ao do Raspberry e com o mesmo SO. Os experimentos foram realizados 50 vezes (IC de 95%) com mensagens de 100Kb e 500Kb tanto no sistema real quando no emulado. Uma solução foi implementada para obter tempo de serialização e deserialização e tempo total das aplicações tanto no cliente como no servidor. Conclui-se que o comportamento no sistema emulado e no sistema real são próximos tanto no cliente como no servidor. No sistema emulado há um leve aumento nos tempos, mas é justificado devido ao custo da emulação.

Crotti et al. (2013) propõe um sistema para acesso remoto utilizando uma placa Raspberry Pi. Foi configurado um servidor *web* na placa que ira comunicar com uma pagina *web* (que deve estar na mesma rede). Através dessa pagina foi possível o gerenciamento de portas (envio de comandos, acionamento e/ou leitura de portas, etc) para o servidor *web* na placa. Também foi configurado uma *webcan* que monitora os experimentos, as imagens são envidas para a placa e esta a envia para a internet (*streaming* de vídeo). Foi configurada na placa: pacote lighttpd (um servidor *web*) para servidor de arquivos em C e também PHP, pacote Moiton para o servidor de câmeras (em que cada câmera recebe uma porta de endereço).

Na pagina *web* foram colocados 3 botões (acender, apagar e verificar o estado de um LED), além de um quadro que exibe o streaming da webcan. Cada um desses botões enviar comandos de leitura de arquivos (.C) específicos localizados no servidor de arquivos. Em cada um desses arquivos contem as rotinas para cada função (acender, apagar e verificar). Concluiu-se que a aplicação proposta obteve ótimos resultados, inclusive da utilização do Raspberry como servidor, o que possibilita a criação de servidores *web* de baixo custo porem com bom desempenho.

Silva e Martins (2012) apresentam uma implementação do Problema do Caixeiro Viajante na estrutura de um AG de forma paralela utilizando a biblioteca OpenMP e a biblioteca Pthreads. Nessa implementação do AG cada combinação de rota (passando por todas as cidades) foi definida como um indivíduo, e um conjunto desses indivíduos uma população; cada nó (ou cidade) foi definido como um gene. Assim quanto maior a população, maior a chance de encontrar uma boa rota porém maior será o processamento necessário para gerar a próxima geração. Foram gerados conjuntos de populações (ou ilhas) que sofrem a mutação um independente da outra (a função que realiza a mutação é executada de forma paralela, ou seja, as ilhas sofrem mutações ao mesmo tempo, porém independentes umas das outras). A implementação foi feita em C e cada indivíduo foi representado por um vetor com tamanho igual ao numero total de cidades (as coordenadas das cidades estavam em um vetor estático de ponto flutuante também com tamanho igual ao numero de cidades) - a população é um vetor de indivíduos. Na implementação usando a biblioteca pthreads, cada *thread* ficou com uma *struct* e nesta havia um vetor de indivíduos (ou população), assim foi possível ter as ilhas isoladas umas das outras (pois cada thread só acessa sua struct).

Os testes foram realizados em dois computadores ambos com a mesma distribuição Linux, um com quatro núcleos reais e outro com dois reais (em que cada núcleo real simula dois núcleos - SMP). Foram ajustados: taxas de erro, numero máximo de gerações, numero de indivíduos em cada população e numero de indivíduos na elite. Os resultados mostram que no computador com SMP a implementação com openMP obteve melhor resultado que a pthread, enquanto que no computador com 4 núcleos reais a pthread obteve melhor resultado. Nos dois casos a implementação paralela obteve resultado significativamente melhor em relação a sequencial. Apesar de que nos cenários propostos a implementação paralela foi melhor que a sequencial, deve-se tomar cuidado ao escolher entre as bibliotecas openMP e pthread, pois cada uma apresenta desempenho melhor em diferentes situações, tornando essencial a análise do problema antes da escolha.

Godói (2015) apresenta uma comparação de custo/benefício entre um cluster de computadores *desktop* tradicionais (computadores HP Compaq 6005 Pro Microtower) e de dispositivos de baixo consumo de energia, como Raspberry Pi Modelo B e Cubietruck. Para testá-los, foram implementados algoritmos (multiplicação de matrizes MxN e Caixeiro Viajante) para resolução de problemas das classes P e NP-Difícil além do *software benchmark* HP Linpack (HPL). A biblioteca MPI foi utilizada para troca de mensagens. Três clusters homogêneos (no formato mestre/escravo) foram montados: computadores desktop, Raspberry Pi e Cubietruck.

Foram realizadas análises para se obter dados de custo energético (foi desenvolvido um sistema com microcontrolador Tiva e um sensor de corrente), financeiro e velocidade de processamento (para esse parâmetro utilizou-se *speedup* e o *benchmark* HPL). Com os dados coletados de ambos os clusters foi possível estimar o custo e benefício gerado, bem como verificar sua aplicabilidade. Concluiu-se que os clusters de Raspberry Pi e de Cunietruck tem custo financeiro e consumo energético muito inferior ao de computadores *desktops*. Porém ao analisar os dados de speedup e do HPL pode-se perceber que a capacidade de processamento do cluster de computadores desktop é muito superior ao demais. Por fim, concluiu-se que cluster de dispositivos ARM são recomendados para aplicações que necessitam de baixo custo e baixo consumo energético, já cluster de computadores desktops são recomendados para aplicações em que o tempo seja crítico ao sistema e não preocupação com custo e consumo (inclusive de refrigeração).

Serckumecka et al. (2015) realizaram a construção de dois clusters isolados um do outro: um com sete computadores HP Compaq 6005 Pro Microtower e outro com oito dispositivos ARM (Raspberry Pi) modelo B. Os computadores com distribuição Linux Ubuntu Server e Raspbian para os dispositivos ARM. A biblioteca MPI foi utilizada para a paralelização. Como parâmetros de análise foram considerados o *benchmark* HP Linpack (HPL), calculo de speedup, consumo de energia e custo monetário dos equipamentos. Os cálculos de speedup serão realizados a partir de algoritmos projetados para solucionar problemas, como: *Quick-Sort*, multiplicação de matriz e Caixeiro Viajante utilizando

o método GRASP (*Greedy Randomized Adaptive Search Procedure*). Para calculo do consumo energético foram utilizados sensores de corrente e tensão.

Conclui-se que é possível utilizar técnicas de paralelização para equilibrar a inferioridade dos dispositivos embarcados em relação aos computadores convencionais. Sendo assim, é possível criar um sistema computacional distribuído de alto poder de processamento e baixo custo, tanto monetário, quanto energético. Contudo os dados obtidos após a avaliação mostram que o cluster de Raspberry Pi é inviável para grandes cargas de processamento e com poucos nós.

Lima (2016) propõe a análise de desempenho de um cluster embarcado de baixo custo composto por processadores da arquitetura ARM e plataforma Raspberry Pi. O trabalho analisa o impacto de usar as bibliotecas MPICH-2 e OpenMPI, executando os programas dos benchmarks HPCC e HPL, a fim de verificar qual implementação obteve um melhor desempenho e um baixo consumo de energia. Os resultados de desempenho e consumo de energia do cluster com esses programas, mostraram que é possível usar clusters de plataformas embarcadas de baixo custo e tendo speedups e consumo de energia satisfatórios.

Este trabalho se difere dos demais ao comparar o desempenho com um computador do tipo *notebook* ao invés do computador de mesa e, não levar em consideração o consumo de energia. Outra diferença está nas métricas de desempenho utilizadas (Sd e Ef, como é apresentado na Seção

4 Materiais e Métodos

Este capítulo apresenta os materiais utilizados para o desenvolvimento, os métodos aplicados e as métricas selecionadas para a análise do desempenho com base nos resultados obtidos.

4.1 Materiais

Para a realização deste trabalho, foi utilizado um notebook Asus X44C com sitema operacional Linux Ubuntu 16.04 professional 64 bits, suas especificações são apresentadas no Quadro 2; uma placa Raspberry Pi 3B com cartão MicroSD de 16GB, suas especificações são apresentadas na Seção 2.5; compiladores GCC e G++ versões 5.4.0; biblioteca OpenMp 3.1; ambiente de desenvolvimento R-Studio 3.3.2; software TeXstudio 2.10.8. O algoritmo escolhido foi um AG paralelo para resolução do Problema do Caixeiro Viajante disponibilizado por Arora (2017).

Item	**Descrição**
Modelo	Notebook Asus X44C
Processador	Intel Core i3-2330M 2.2GHz
Memóra	4 GB DDR3 1333MHz SDRAM *On-board*
Sistema Operacional	Linux Ubuntu 16.04 LTS - 64 *bits*
Armazenamento	HD SATA 500 GB - 5400 rpm
Fonte de Alimentação	Input: 127/220 V AC, 50/60Hz
	Output: 19V DC, 3,42A, 65W

Quadro 2 – Especificações de *hardware* (processador, memória, HD, alimentação, etc.) do *Notebook* Asus X44C usado no desenvolvimento deste trabalho.
Fonte: (ASUSTEK, 2017).

O arquivos de entrada para o algoritmo, possuem três colunas. A primeira é um número inteiro que representa a cidade, a segunda e a terceira são respectivamente a coordenada X e Y das cidades (seno números reais entre 0 e 10000 gerados aleatoriamente). O algoritmo primeiramente armazena o tempo atual, após isso realiza a leitura do arquivo de entrada e cria uma matriz com as coordenadas X e Y das cidades. Em seguida é gerada uma matriz de distância das cidades. Esta é a distância euclidiana, que em um espaço bi-dimensional (entre os pontos $P = (p_1, p_2, ..., p_n) e Q = (q_1, q_2, ..., q_n)$) é dada pela Equação 1.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad (1)$$

Na sequência é gerada aleatoriamente a população inicial. O máximo de gerações é de 100000, sendo que nas 8000 (80%) primeiras gerações a taxa de mutação é de 20% e nas 2000 (20%) restantes a taxa de mutação aumenta para 40%. O algoritmo armazena também os 10 melhores cromossomos (impedindo a extinção dos mesmos) de todas as gerações para ressoá-los de tempos em tempos. O *fitness* é avaliado a cada geração, inclusive o da população inicial. Há disponíveis 4 métodos diferentes de cruzamento dos cromossomos - *Partially Mapped Crossover* (PMX), *Greedy Crossover* (GX), *Cycle Crossover* (CX) e *Edge Recombination Crossover* (ERX) - e a cada geração é sorteado aleatoriamente um destes métodos.

4.2 Métricas

Esta seção apresenta as métricas que foram aplicadas nos dados obtidos após a execução do AG para avaliação dos resultadas.

1. ***Speedup (Sd)***: é uma métrica de desempenho utilizada para avaliar o quanto um algoritmo, uma execução ou arquitetutra é mais rápida que a outra. Um exemplo seria a relação entre o tempo sequencial e o tempo paralelo de um mesmo algoritmo. A Equação 2 apresenta o modelo matemático do *Sd*. (RODRIGUES, 2012).

$$Sd = \frac{T_s}{T_p} \tag{2}$$

   Onde T_s é o tempo sequencial e T_p é o tempo paralelo. Esta equação representa um fator de ganho, caso seu resultado seja um valor maior ou igual a 1, o algoritmo paralelo é mais rápido que o sequencial, caso contrário significa que o sequencial é mais rápido.

2. **Eficiência (Ef)**: representa a eficiência obtida pelo processamento paralelo em relação a quantidade de núcleos disponíveis. Seu modelo matemático é apresentado na Equação 3. (RODRIGUES, 2012).

$$Ef = 100 * (\frac{Sd}{N\acute{u}cleos}) \tag{3}$$

   Este resultado mostra o quanto o algoritmo paralelo utilizou dos núcleos (utilização dos núcleos em relação ao ganho). O valor ideal de eficiência seria de 100%, mas na prática isso não ocorre, devido a várias questões como estratégia de escalonamento de processos, sincronização de dados, liberação de recursos, entre outros.

3. **Média aritmética**: é a soma total ($\sum$) dos dados divididos pela quantidade de dados (n) e é usada para resumir dados quantitativos simétricos. Considerando que $x_1, x_2, ..., x_n$ são os valores dos dados, a Equação 4 apresenta o modelo matemático

da média. Alguns autores consideram essa uma medida de tendência central devido ao fato de focar nos valores médios dentro da amostra (entre os maiores e os menores). (SHIMAKURA, 2005).

$$\bar{x} = \frac{x_1 + x_2 + ... + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4}$$

4. **Mediana**: ou (*percentil* 50) é uma medida de localização do centro da distribuição dos dados. É dada pelo valor que, ao ordenar os valores dos dados da amostra, divide os mesmos ao meio e com isso metade dos dados tem valores menores que a mediana e a outra metade tem valores maiores que a media, sendo principalmente útil em dados não simétricos. Para uma amostra par a mediana é a média dos dois valores centrais, para amostra ímpar é obtida pela Equação 5. (SHIMAKURA, 2005).

$$Md = \frac{n+1}{2} \tag{5}$$

5. **Desvio Padrão**: para se obter o desvio padrão, faz-se necessário calcular a variância - definida como o desvio quadrático médio da média - dada pela Equação 6. O mesmo é dados pela Equação 7.

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} = \frac{\sum_{i=1}^{n}(x_i)^2 - n\bar{x}^2}{(n-1)} \tag{6}$$

$$\sigma = \sqrt{variância} = \sqrt{s^2} \tag{7}$$

O desvio padrão é uma forma de expressar a variabilidade dos dados eliminando a influência da ordem de grandeza da variável. Shimakura (2005) comenta que o desvio padrão:

a) analisado como a variabilidade dos dados em relação à média. Quanto menor for, mais homogêneo é a amostra;

b) adimensional (positivo quando a média for positiva e zero quando não houver variabilidade no conjunto de dados);

c) para qualquer conjunto de dados, pelo menos 75% deles ficam dentro de uma distância de 2 desvios padrão da média, isto é, entre $\bar{x} - 2s$ e $\bar{x} + 2s$.

4.3 Métodos

Os testes foram realizados com entradas (cidades) de tamanho 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95 e 100, sendo executadas de forma paralela e sequencial em ambas as arquiteturas apresentadas, a fim de verificar o comportamento das

mesmas com diferentes volumes de dados. Um segundo teste com entrada de tamanho 30 foi realizado com 50 repetições, gerando uma amostra, em cada um dos cenários apresentados para analisar o comportamento das arquiteturas ao longo do tempo. Os tempos de cada execução foram fornecidos pelo próprio algoritmo, os mesmos foram armazenados para aplicação das métricas apresentadas na Seção 4.2.

5 Resultados de Discussões

Após a realização do primeiro teste, que variava o tamanho da entrada, foi possível analisar o comportamento de cada arquitetura tanto no AG paralelo quato no sequencial. A Figura 6 mostra o gráfico obtido do AG sequencial e a Figura 7 mostra o gráfico obtido do AG paralelo executados no *notebook*.

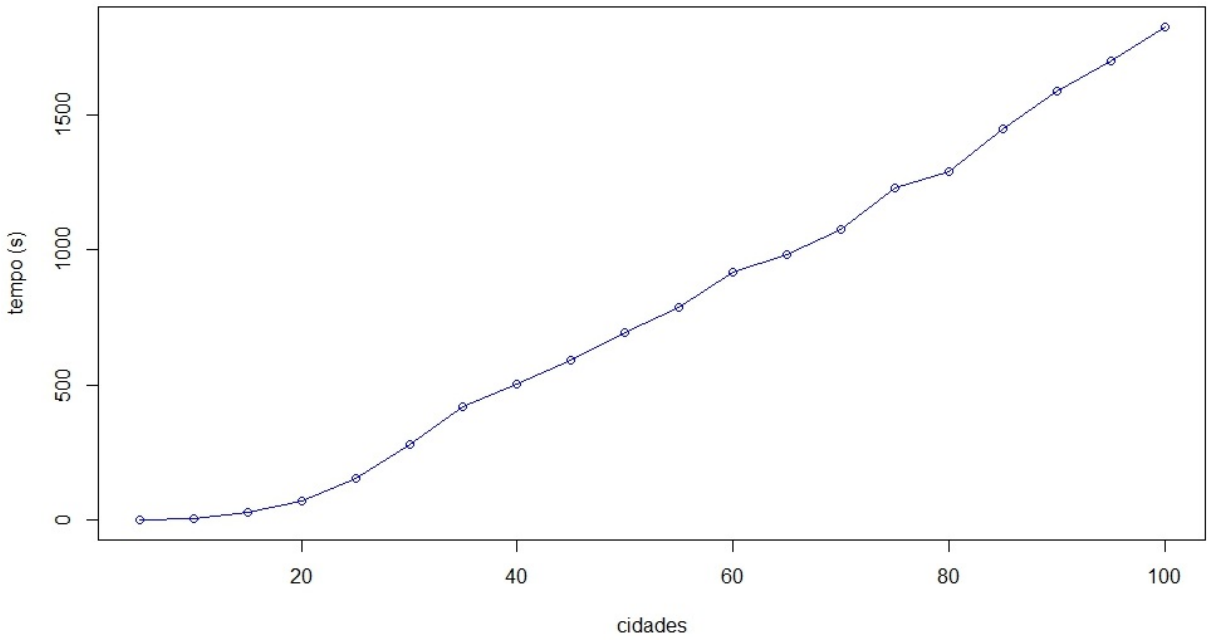

Figura 6 – *Notebook*: tempo sequencial.
Tempo em função da quantidade de cidades no algoritmo sequencial.

As Figuras 8 e 9 mostram, respectivamente, os gráficos obtidos do AG sequencial e paralelo executados na placa Raspberry Pi.

Observando-se as Figuras 6, 7, 8 e 9, pode-se perceber que as arquiteturas apresentam comportamento semelhantes no AG tanto sequencial quanto no paralelo, o que diferencia é a escala de tempo, já que em cada cenário apresentou faixas de tempo diferentes.

Com base nos tempos adiquiridos foram calculados o Sd e a Ef, que são mostrados na Tabela 2. É possível perceber o Sd do *notebook* em relação a Raspberry Pi (no AG sequencial e no paralelo) é de aproximadamente 3,3, o que significa que o computador é 3,3 vezes mais rápido que a placa, em ambos os cenários. Ao analisar o Sd do Tp em relação ao Ts do *notebook*, nota-se que o paralelo é quase duas vezes mais rápido que o sequencial. Já no Sd do Tp em relação ao Ts da placa Raspberry Pi, o paralelo é aproximadamente 1,7 vezes mais rápido que o sequencial. As duas arquiteturas apresentaram valores bem próximos de Ef, sendo aproximadamente 50% para o *notebook* e 48% para a Raspberry Pi.

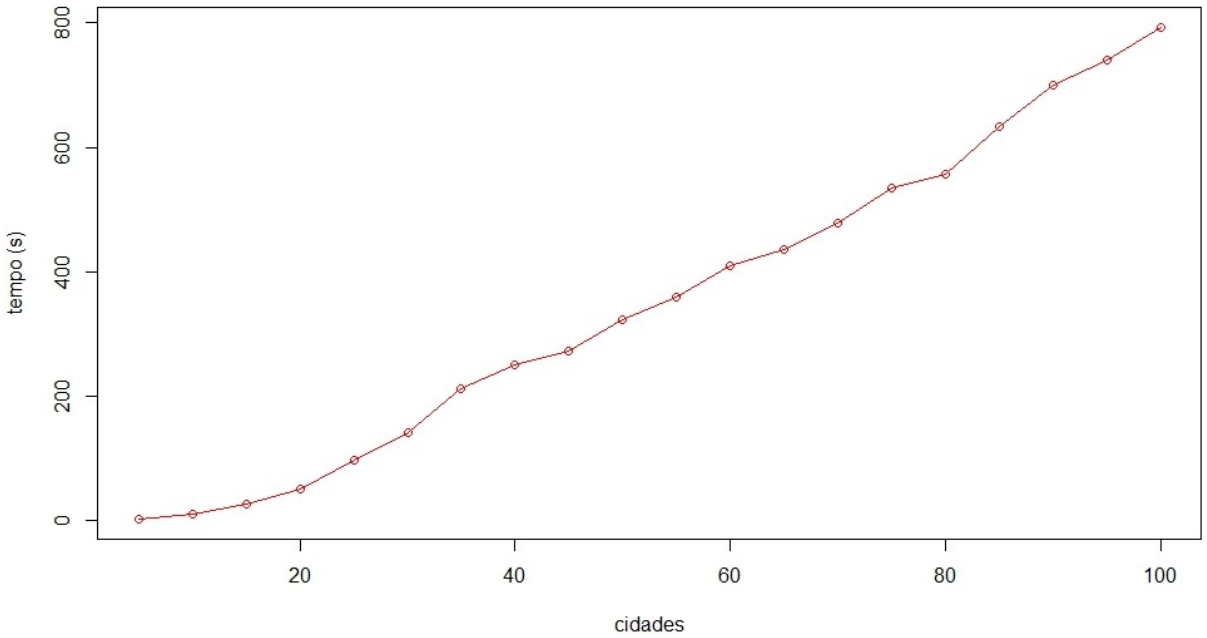

Figura 7 – *Notebook*: tempo paralelo.
Tempo em função da quantidade de cidades no algoritmo paralelo.

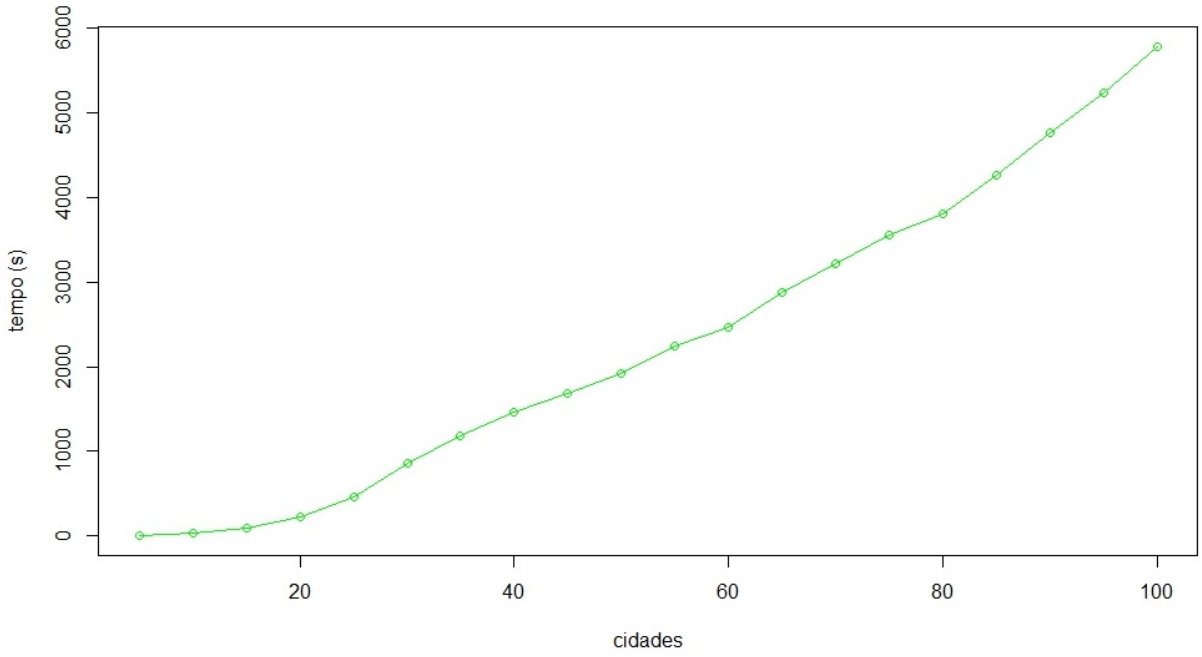

Figura 8 – Raspberry Pi: tempo sequencial.
Tempo em função da quantidade de cidades no algoritmo sequencial.

As Figuras 10 e 11 mostram, respectivamente, o comportamento da Ef das arquiteturas conforme a quantidade de cidades no arquuivo de entrada. Ao observá-las, fica evidente que a Ef melhora com o aumento da carga de processamento. Isso se deve ao fato de que para pequenas cargas, o custo para divisão das tarefas (alocação e escalonamento de *threads*, por exemplo) reduz a eficiência (aumenta o tempo da execução paralela) das arquiteturas.

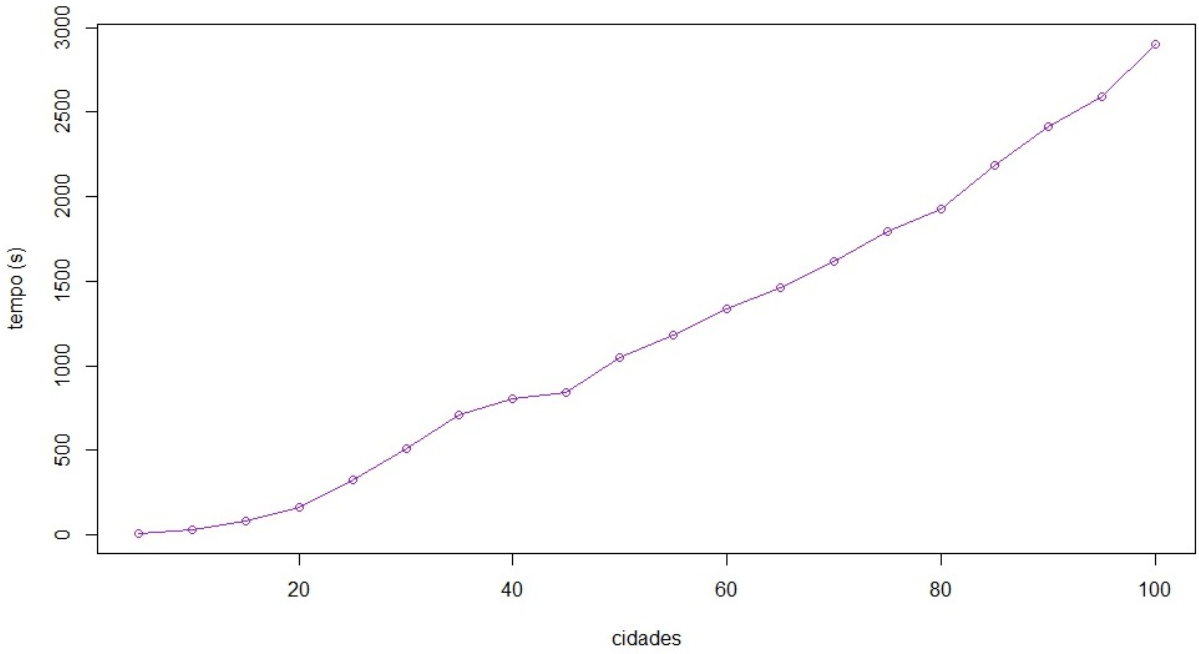

Figura 9 – Raspberry Pi: tempo paralelo.
Tempo em função da quantidade de cidades no algoritmo paralelo.

Tabela 2 – Resultados - que incluem a media, mediana e desvio padrão - obtidos a parir do teste realizado com 20 repetições de execução, variando o tamanho das entradas, em cada um dos cenários propostos.

Parâmetro analisado	**Media**	**Mediana**	**Desvio Padrão**
***Sd Ts* Raspberry/Notebook**	3,2695	2,9558	1,0925
***Sd Tp* Raspberry/Notebook**	3,3224	3,3456	0,1667
Sd* Notebook *Ts/Tp	1,9123	2,1792	0,5637
Sd* Raspberry *Ts/Tp	1,7350	1,8731	0,3314
***Ef* Notebook**	50,1549	54,4800	22,9936
***Ef* Raspberry**	47,9383	48,1387	21,9332

Ao analisar a Figura 12 pode-se perceber a distribuição das medianas dos Sd_s obtidos no primeiro teste, onde é possível notar alguns valores discrepantes (ou *outliers*) no Sd do *notebook* em relação a Raspberry Pi, o que justifica o desvio padrão (Tabela 2) um pouco maior que os demais Sd_s apresentados. Esses valores são resultados de execuções anormais se comparadas com a maioria e, mesmo sendo *outliers*, são normais e esperados em sistemas de propósito geral, visto que estes executam outras aplicações ao mesmo tempo, fazendo com que vários fatores interfiram no tempo total de execução.

No segundo teste, foram repetidas 50 execuções do AG sequencial e paralelo nas duas arquiteturas, sem variar a quantidade de cidades, que foi fixada em 30. Dessa forma foi possível analisar o comportamento das arquiteturas ao longo do tempo. As Figuras 13 e 14 apresentam as frequências (histograma) de Ts_s, Tp_s e Sd_s obtidos no segundo teste.

Ao analisar a frequência dos tempos de execução, Figura 13, pode-se observar que as arquiteturas apresentaram certa estabilidade nas execuções, visto que ouve pouca

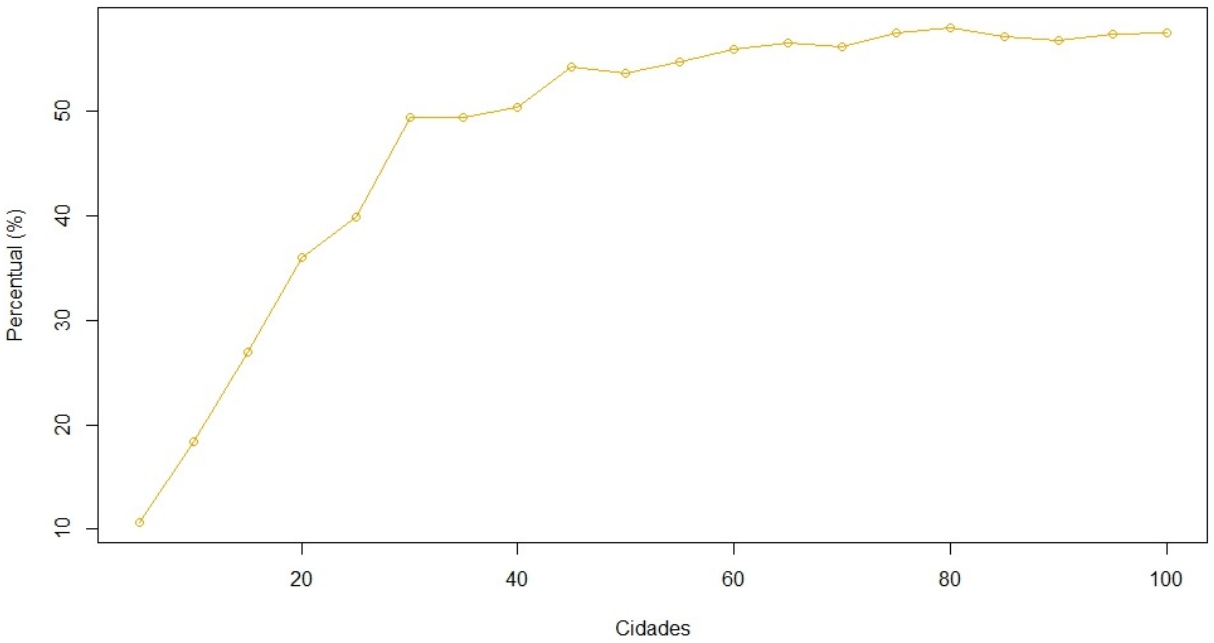

Figura 10 – *Notebook*: eficiência.
Comportamente do eficiência em função do tamanho das entradas (quantidade de cidades).

varição dos tempos nas faixas de maior frequência. A mesma situação ocorre na Figura 14 (frequências altas em faixas pequenas), com os valores de Sd_s. A execução sequencial na Raspberry Pi foi a que apresentou maior variabilidade nos tempos. Todos os cenários apresentam alguns valores discrepantes, porém os mesmos são esperados e considerados

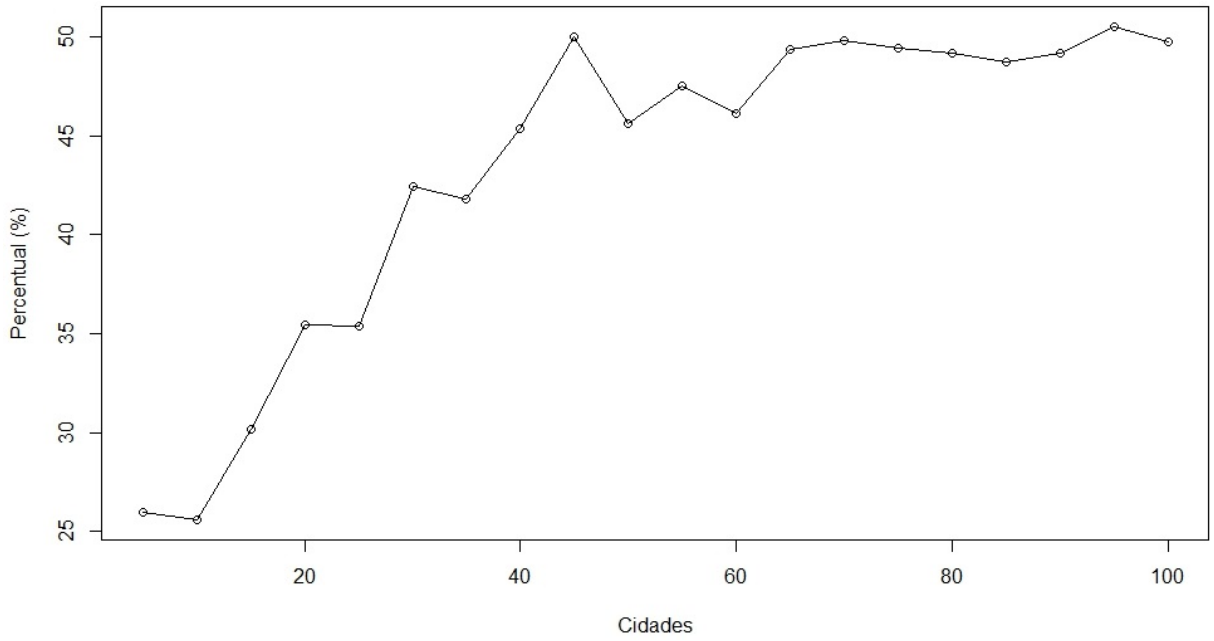

Figura 11 – Raspberry Pi: eficiência.
Comportamente do eficiência em função do tamanho das entradas (quantidade de cidades).

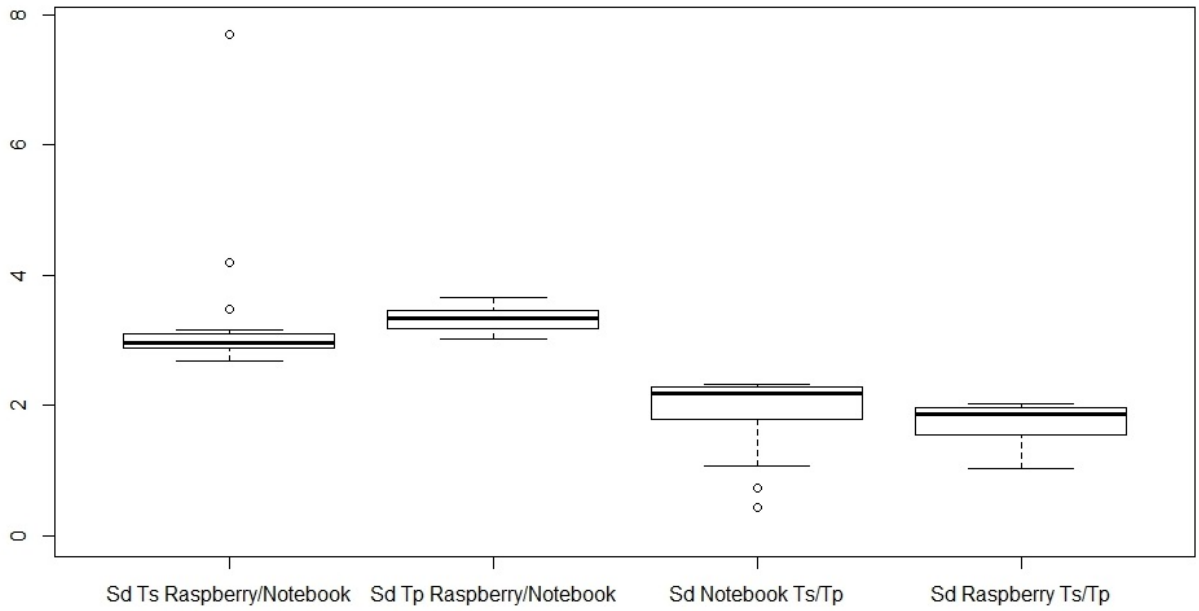

Figura 12 – Comparação dos Sd_s obtidos no primeiro teste.
O gráfico Boxplot mostra a comparação entre as medianas dos Sd_s obtidos, além de suas variâncias.

normais em arquiteturas de propósito geral. Na Figura 15 são apresentados os gráficos boxplot dos Sd_s calculados, que mostram resultados levemente diferentes aos obtidos no primeiro teste: o *notebook* 4,8 vezes mais rápido que a Raspberry Pi no Ts e 3,4 vezes mais rápido no tempo paralelo, o AG paralelo é 1,9 vezes mais rápido que o sequencial no *notebook* e na Raspberry Pi é 2,8 vezes mais rápido. Na Figura 15 também é possível observar os *outliers* presentes nos Sd_s.

A Tabela 3 mostra a média, mediana e desvio padrão dos tempos de execução sequenciais e paralelos, Sd_s e Ef das arquiteturas analisadas.

Tabela 3 – Resultados - que incluem a media, mediana e desvio padrão - obtidos a parir do teste realizado com 50 repetições de execuções, com entrada de tamanho fixo, em cada um dos cenários propostos.

Parâmetro analisado	**Media**	**Mediana**	**Desvio Padrão**
***Ts* Notebook**	279,7962	279,7430	0,5213
***Ts* Raspberry**	1370,545	1373,180	7,0393
***Tp* Notebook**	139,5995	139,5775	0,3903
***Tp* Raspberry**	481,2862	481,3365	1,5111
***Sd Ts* Raspberry/Notebook**	4,8897	4,8995	0,0381
***Sd Tp* Raspberry/Notebook**	3,4266	3,4439	0,0785
Sd* Notebook *Ts/Tp	1,9936	2,0032	0,0456
Sd* Raspberry *Ts/Tp	2,8449	2,8466	0,0213
***Ef* Notebook**	49,8415	50,081	1,1407
***Ef* Raspberry**	71,1238	71,165	0,5339

As Figuras 16 e 17 mostram o comportamento, respectivamente, das Ef_s do *notebook*

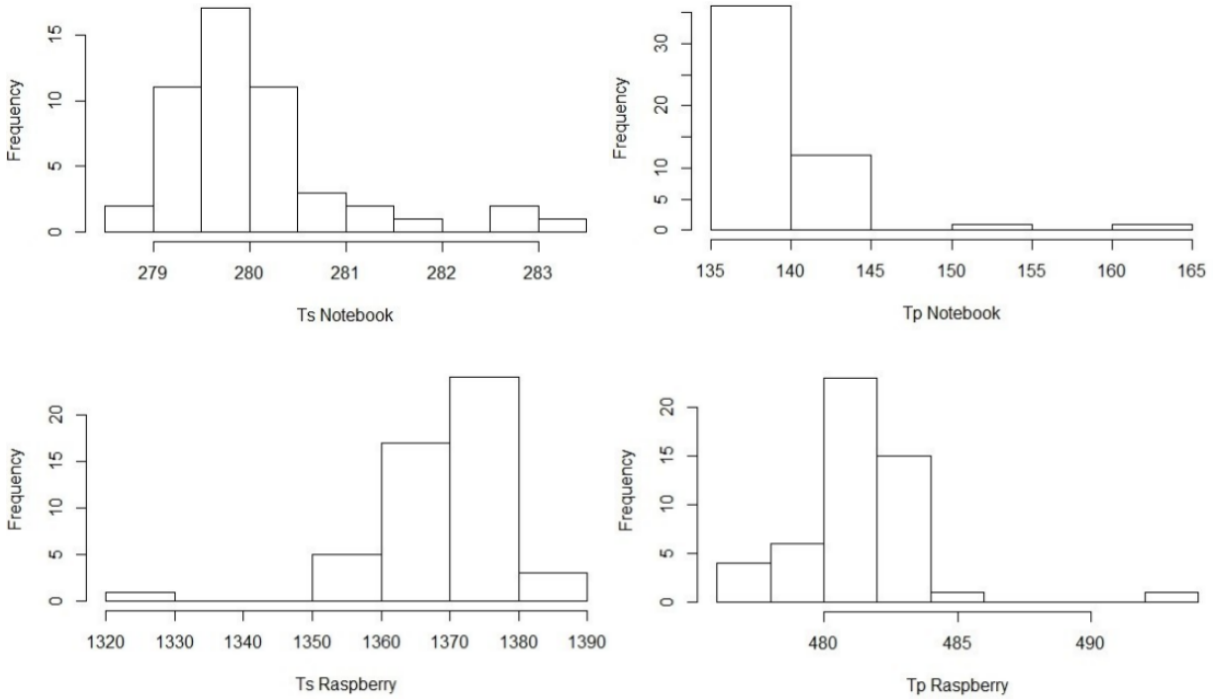

Figura 13 – Histogramas dos tempos de execução obtidos.
Os gráficos mostram a frequência das ocorrências dos tempos de execução do AG, em todos os cenários, obtidos no segundo teste.

e da Raspberry Pi no decorrer da execuções. Com base nestas imagens e nos valores de Ef apresentados na Tabela 3, fica claro que a Raspberry obteve uma eficiência melhor se comparada ao *notebook*. Isso se deve principalmente as otmizações a nível de SO da placa, por ser um sistema mais limitado em relação ao *notebook*, que permitiram um melhor aproveitamento dos núcleos.

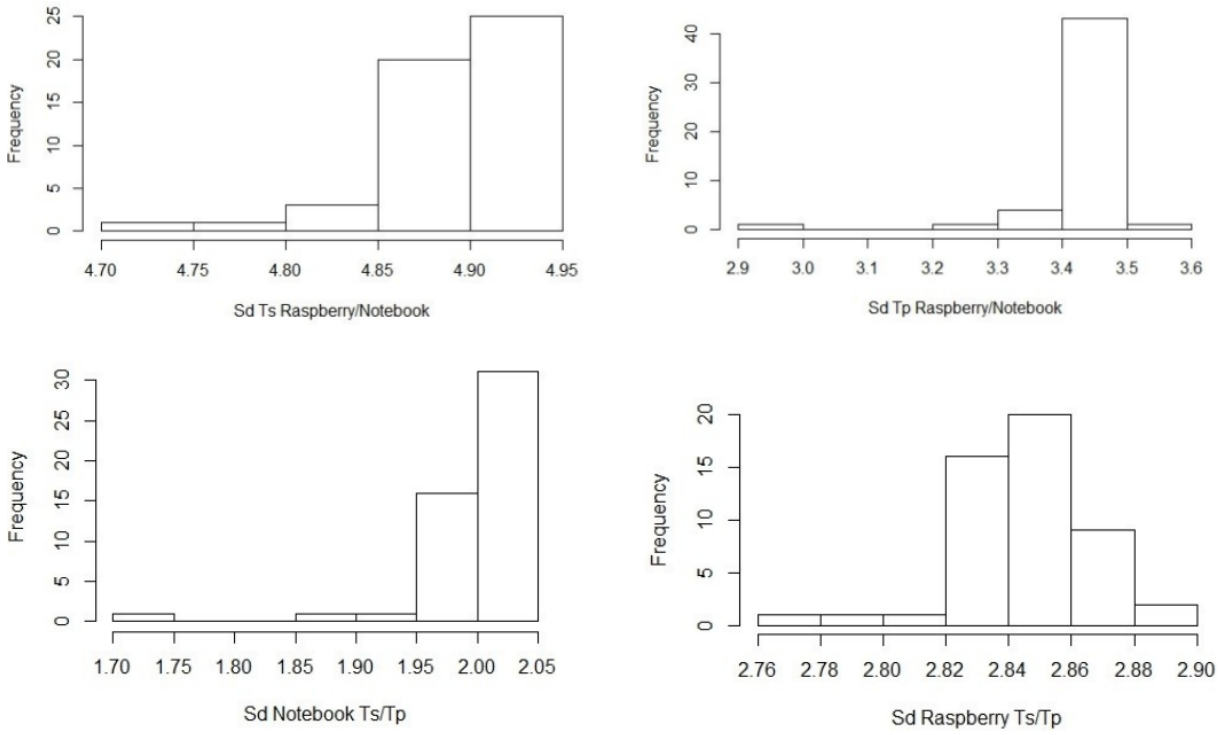

Figura 14 – Histogramas dos Sd_s obtidos.
Os gráficos mostram a frequência dos Sd_s ocorridos nos diferentes cenários propostos, obtidos no segundo teste.

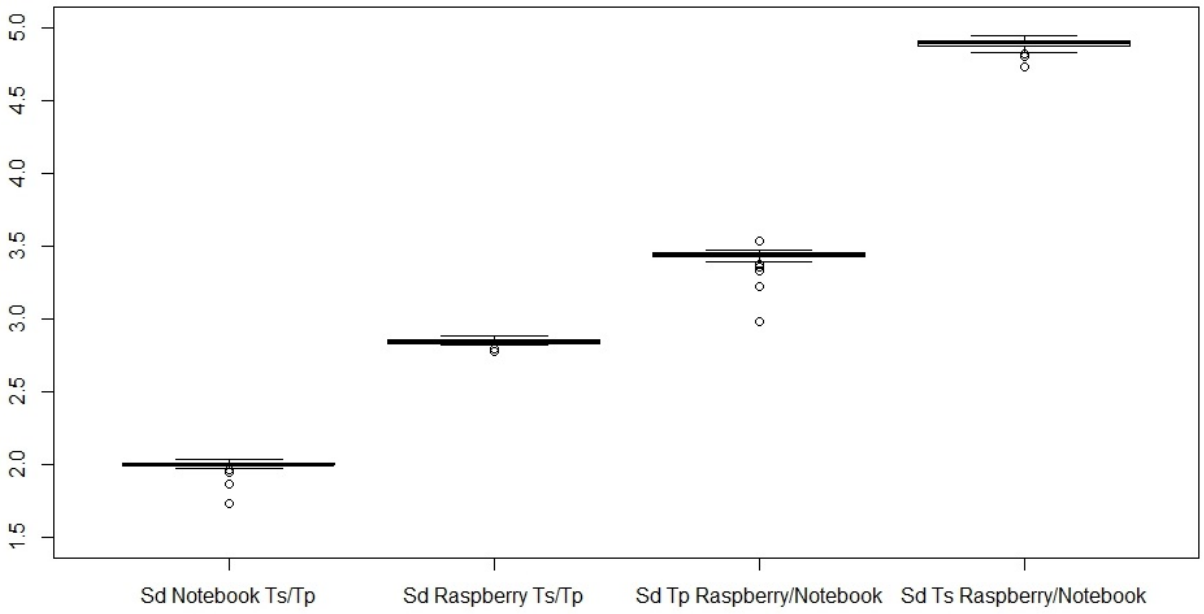

Figura 15 – Comparação dos Sd_s obtidos no segundo teste.
O gráfico Boxplot mostra a comparação entre as medianas dos Sd_s obtidos, além de suas variâncias.

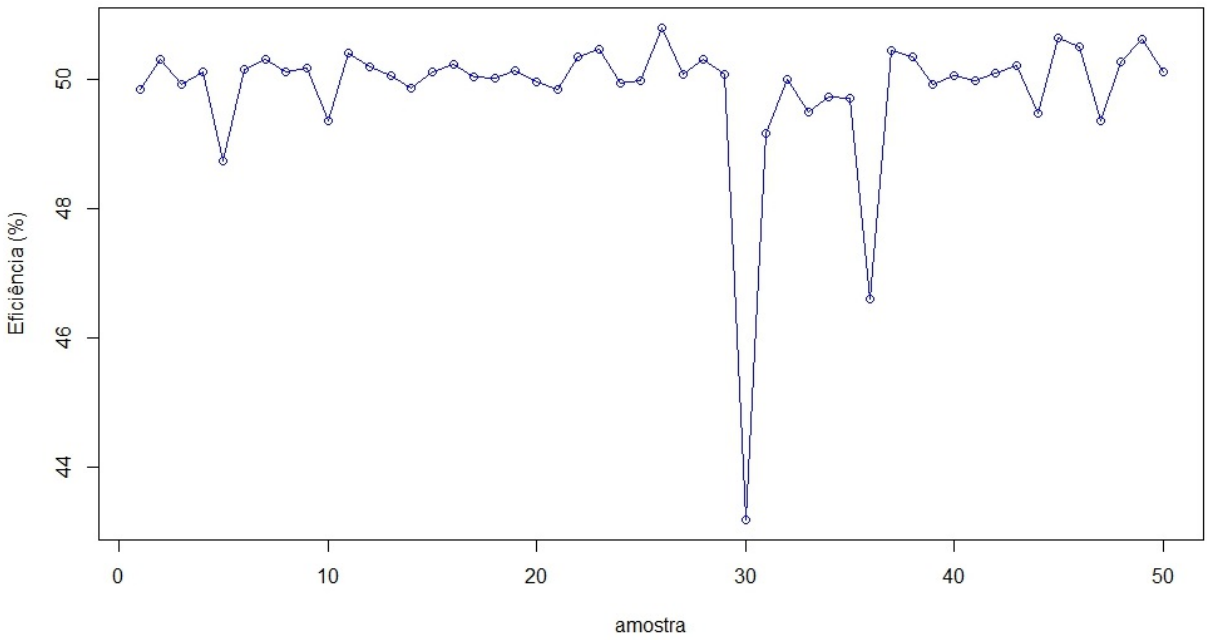

Figura 16 – *Notebook*: eficiência.
Comportamento da Ef do *notebook* ao longo das execuções com entradas do mesmo tamanho.

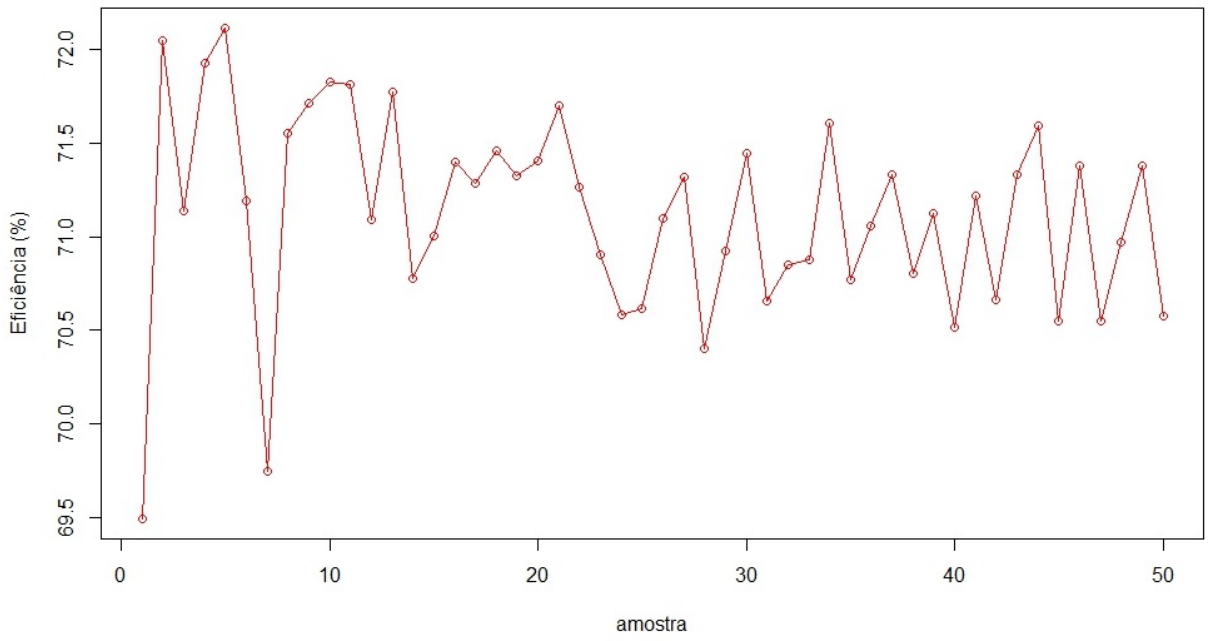

Figura 17 – Raspberry Pi: eficiência.
Comportamento da Ef da placa Raspberry Pi ao longo das execuções com entradas do mesmo tamanho.

6 Conclusões

Neste trabalho foi realizado um comparativo de desempenho entre um computador *desktop* e uma Raspberry Pi. Vale ressaltar que o objetivo não é testar qual arquitetura é melhor em relação a outra, mas sim quão bom pode ser o desempenho de uma arquitetura embarcada se comparada ao de um computador comum. Para a realização dos testes, um AG para resolução do PCV foi utilizado de forma sequencial e paralela (uso da biblioteca OpenMP). Foram realizados dois testes, sendo que o primeiro era a execução do AG com entradas de tamanhos diferentes e o segundo execuções repetidas (total de 50) com tamanho de entrada fixo (30 cidades) em ambas as arquiteturas propostas. Com o tempo de execução obtido foram aplicadas as métricas como Sd e Ef, entre outras, para análise do desempenho e comportamento das arquiteturas.

Os resultados mostram um Sd de 3 a 4,8 do *notebook* em relação a Raspberry Pi no AG sequencial e um Sd em torno de 3,3 do *notebook* em relação a Raspberry Pi no AG paralelo. Estes resultados são esperados visto que a placa possui recursos mais limitados em relação ao *notebook*, dos quais podemos destacar:

1. Memória secundária: o armazenamento da Raspberry Pi é um cartão MicroSD (memória *flash*) que é mais lenta se comparada a do *notebook* (HD SATA);

2. Memória primária: a memória da Raspberry Pi (DDR2 900 MHz) possui frequência menor que a do *notebook* (DDR3 1,33GHz);

3. Arquitetura x86 (híbrida de RISC e CISC) do *notebook* possui instruções complexas já implementadas, enquanto que a arquitetura ARM (puramente RISC) não tem, o que aumenta a quantidade de instruções necessárias para executar determinadas tarfeas, se comparada com a x86.

Por outro lado, a placa Raspberry Pi apresentou melhor eficiência na utilização de seus núcleos quando comparada com o *notebook*. Isso se deve principalmente a otimização e robustez presentes da placa devido ao fato de ter seus recursos mais limitados o que inclui um SO mais enchuto.

Por fim, pode se concluir que a placa Raspberry Pi possui desempenho sigfificativo que, aliado a outras vantagens (baixo custo, estabilidade, recursos como porta *ethernet*, wi-fi, GPIO, entre outros), torna a placa uma alternativa barata e eficiente para aplicações de baixo custo e que não necessitam de tempo de resposta como fator crítico.

6.1 Limitações

Pode-se destar como principais limitações a falta de um ambiente de rede (*cluster*) para testes com aplicações distribuídas, falta de testes com outros tipos de aplicações que exijam carga de processamento e análises utilizando *softwares* do tipo *benchmark*. O consumo de energia e o desempenho com e sem um dissipador de calor para a Raspberry Pi também não foram analisados.

6.2 Trabalhos futuros

Como sugestão para trabalhos futuros:

1. Testes com aplicações distribuídas utilizando *cluster* homogêneo;
2. Testes com aplicações distribuídas utilizando *cluster* heterogêneo;
3. Comparação com dispositivos similares a Raspberry Pi, como emxemplo, a Orange Pi e a BeagleBone, entre outras;
4. Testes com outras classes de algoritmos, por exemplo, para processamento gráfico, processamento de digital de imagens (PDI), mineração de dados em grandes volumes de informações, entre outras;
5. Análise do consumo de energia, calor dissipado e a difernça de desempenho com e sem um dissipador de calor para a placa.

Referências

ARM-LTD. Armv8-a architecture. *The Architecture for the Digital World*, 2017. Disponível em: <https://www.arm.com/products/processors/instruction-set-architectures/armv8-architecture.php>.

ARORA, R. Travelling salesman openmp. *GitHub, Inc.*, 2017. Disponível em: <https://github.com/rachit95arora/travelling-salesman-openmp>.

ASUSTEK, C. I. X44c. *asus.com*, 2017. Disponível em: <https://www.asus.com/br/Laptops/X44C/specifications/>.

AZEVEDO, D. L. O. d.; OLIVEIRA, N. M. d. Comparação entre as arquiteturas risc e cisc. *Universidade Federal Rural de Pernambuco - UFRPE*, p. 2, 2014. Disponível em: <http://deinfo.ufrpe.br/14064/arttigos/pt-br/comparaç~ao-entre-arquiteturas-risc-e-cisc>.

COUTINHO, M. P. *Sistema de monitoramento residencial*. Brasília, DF: [s.n.], 2016. 58 p. Disponível em: <http://repositorio.uniceub.br/handle/235/8702>.

CROTTI, Y. et al. Raspberry pi e experimentação remota. *Araranguá, SC*, 2013.

CRUZJúNIOR, S. C. d. *Desenvolvimento de uma plataforma elaborada para projetos de sistemas embarcados reconfiguráveis (ARM7 e FPGA)*. 234 p. Dissertação (Mestrado) — Universidade de Brasília, Brasília, DF, 2013. Disponível em: <http://repositorio.unb.br/handle/10482/12026>.

CUNHA, C. B. d.; BONASSER, U. d. O.; ABRAHãO, F. T. M. Experimentos computacionais com heurísticas de melhorias para o problema do caixeiro viajante. In: . [S.l.: s.n.], 2002.

FULLER, S. H.; MILLETT, L. I. Computing performance: Game over or next level? *Computer*, v. 44, n. 1, p. 31–38, 2011. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/MC.2011.15>.

FUNDAçãORASPBERRYPI. Raspberry pi 3 model b. *Raspberry Pi Blog*, 2017. Disponível em: <https://www.raspberrypi.org/documentation>.

GODóI, F. N. d. *Estudo Comparativo entre Clusters de Computadores Desktop e de Dispositivos ARM*. Pato Branco, PR: [s.n.], 2015. 95 p. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/5707>.

GOLDBERG, D. E. Genetic algorithms in search, optimization, and machine learning. *Reading: Addison-Wesley*, New York, 1989.

HELSGAUN, K. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Jour. of Operational Research*, Elsevier, v. 126, n. 1, p. 106–130, 2000.

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: MIT press, 1992.

LAPORTE, G. et al. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, Wiley Online Library, v. 7, n. 4-5, p. 285–300, 2000.

LIMA, F. d. A. *Implantação e análise de desempenho de um cluster com processadores ARM e plataforma raspberry Pi*. 66 p. Dissertação (Mestrado) — Universidade Federal de Sergipe, São Cristóvão, SE, 2016. Disponível em: <https://bdtd.ufs.br/handle/tede/3325>.

LORENZONI, R. K. *Análise de desempenho e consumo energético entre processadores ARM E x86*. Ijuí, RS: [s.n.], 2012. 81 p. Disponível em: <http://bibliodigital.unijui.edu.br:8080/xmlui/handle/123456789/1060>.

MANO, M. M. Computer system architecture. Prentice Hall, Englewood Cliffs, New Jersey, EUA, 1993.

MARMITT, G. P. *Análise de desempenho com a paralelização do Cálculo de números perfeitos em arquitetura ARM, X86 e INTEL XEON PHI*. Ijuí, RS: [s.n.], 2017. 52 p. Disponível em: <http://bibliodigital.unijui.edu.br:8080/xmlui/handle/123456789/4599>.

MOTYCZKA, L. B. et al. Um comparativo de consumo elétrico entre arquiteturas x86 e arm em servidores de bancos de dados. p. 6, 2013. Disponível em: <https://www.researchgate.net/publication/259339458>.

NAVAUX, P. O. A.; ROSE, C. A. F. D.; PILLA, L. L. Fundamentos das arquiteturas para processamento paralelo e distribuído. *XI Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul - Porto Alegre, RS*, p. 22–59, 2011. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/erad-rs/2011/003.pdf>.

NETO, S. P. Computação evolutiva: Desvendando os algoritmos genéticos. *Faculdade de Jaguariúna, Universidade São Francisco*, v. 1, n. 1, p. 34, 2011.

NEUMANN, J. V. First draft of a report on the edvac. *Moore School of Electrical Engineering, University of Pennsylvania*, IEEE, 1945.

NUNES, L. et al. Análise de desempenho em dispositivos limitados e emulados. estudo de caso: Raspberry pie web services restful. In: *WPerformance-XIII Workshop em desempenho de Sistemas Computacionais e de Comunicação*. [s.n.], 2014. p. 5. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wperformance/2014/021.pdf>.

PACHECO, M. A. C. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada*, p. 9, 1999. Disponível em: <http://www2.ica.ele.puc-rio.br/Downloads\%5C38/CE-Apostila-Comp-Evol.pdf>.

PAPPA, G. L. Conceitos e aplicações em aprendizado de máquina. *Curso Verão*, 2015. Disponível em: <http://homepages.dcc.ufmg.br/~glpappa/cverao/CursoVerao-Parte1.pdf>.

RAMOS, R. M.; RALHA, C.; TEODORO, G. Avaliação de cluster raspberry pi para execução de aplicações de análises de imagens microscópicas médicas. *Brasília, DF*, 2016.

REINELT, G. *The traveling salesman: computational solutions for TSP applications*. [S.l.]: Springer-Verlag, 1994.

RODRIGUES, L. M. *Proposta e Avaliação do Algoritmo K-Means Paralelo e Distribuído Para Predição do Sítio de Início de Tradução em RNA Mensageiro Para Identificação de Proteínas*. 94 p. Dissertação (Mestrado) — Pontifícia Universidade Católica de Minas Gerais (PUC), Belo Horizonte, MG, 2012. Disponível em: <http://homepages.dcc.ufmg.br/~laerte/public/docs/dissertacao.pdf>.

ROSSI, A. L. D. *Ajuste de parâmetros de técnicas de classificação por algoritmos bioinspirados*. São Carlos, SP: [s.n.], 2009. 149 p. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06052009-114528>.

SERCKUMECKA, A. et al. Avaliação do uso da computação paralela em redes de computadores desktop e dispositivos arm. *Anais: Computer on the Beach*, p. 529–531, 2015. Disponível em: <https://siaiap32.univali.br//seer/index.php/acotb/article/view/7115>.

SHEIDEMANTEL, F. L. *Monitoramento de vídeo por meio do computador Raspberry Pi*. Brasília, DF: [s.n.], 2015. 56 p. Disponível em: <http://bdm.unb.br/handle/10483/14756>.

SHIMAKURA, S. Dados quantitativos. *Laboratório de Estatística e Geoinformação*, 2005.

SILVA, E. E. d. *Otimização de estruturas de concreto armado utilizando algoritmos genéticos*. 194 p. Dissertação (Mestrado) — Universidade de São Paulo, Escola Politécnica, São Paulo, SP, 2001. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3144/tde-21022002-112505/en.php>.

SILVA, H. H.; MARTINS, C. A. P. S. Avaliação de implementações do algoritmo genético paralelo para solução do problema do caixeiro viajante usando openmp e pthreads. *XIII Simpósio em Sistemas Computacionais WSCAD-SSC - Workshop de Iniciação Científica*, 2012. Disponível em: <https://s3.amazonaws.com/academia.edu.documents/30275596/Artigo.pdf>.

SILVA, L. F.; ANTUNES, V. J. M. Comparação entre as arquitecturas de processadores risc e cisc. *Cidade do Porto, Portugal*, p. 9, 2008. Disponível em: <http://www.inf.unioeste.br/~guilherme/oac/Risc-Cisc.pdf>.

STALLINGS, W. *Arquitetura e organização de computadores: projeto para o desempenho*. 5. ed. Prentice Hall, 2002. ISBN 85.87918-53-2. Disponível em: <http://www.ncdd.com.br/livros/Arquitetura-e-Organizacao-de-Com-Put-Adores-5-Ed-William-Stallings.pdf>.

WEI, H. et al. Research on reconfigurable robot controller based on arm and fpga. In: IEEE. *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. [S.l.], 2008. p. 123–128.

ÁVILA, S. L. *Algoritmos genéticos aplicados na otimização de antenas refletoras*. 98 p. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Centro Tecnológico, Florianópolis, SC, 2002. Disponível em: <https://repositorio.ufsc.br/handle/123456789/84439>.