HORUS: Servidor de Imagens com Suporte as Unidades CSD

Luiz Márcio Faria de Aquino Viana, M. Sc.

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS DE COMPUTAÇÃO.

Aprovada por:

_____
Prof. Felipe Maia Galvão França, Ph. D.

_____
Prof. Diego Cadette Dutra, D. Sc.

_____
Prof. Tiago Alves, D. Sc.

_____
Prof[a]. Priscila Machado Vieira Lima, Ph.D.

_____
Prof. Claudio Miceli de Farias, D. Sc.

_____
Prof. Alexandre de Assis Bento Lima, D. Sc.

_____
Prof. Guilherme Horta Travassos, D. Sc.

RIO DE JANEIRO, RJ - BRASIL
NOVEMBRO DE 2022

VIANA, LUIZ MÁRCIO FARIA DE AQUINO

HORUS: Servidor de Imagens com Suporte as Unidades CSD [Rio de Janeiro] 2022

XII, [NUM_PAGINAS] p. 29,7 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 2022)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Armazenamento de Dados Distribuído
2. Unidades de Armazenamento
3. Rede de Sensores
4. Microcontrolador
5. Arquitetura de Processador

I. COPPE/UFRJ II. Título (série)

DEDICATÓRIA

Aos meus filhos, Luiz Felipe e Maria Júlia,
por toda a alegria e força que me proporcionam.

AGRADECIMENTOS

Agradeço a Deus pela força e luz recebida durante o desenvolvimento deste trabalho e pela oportunidade que eu tive em ampliar o meu conhecimento.

Aos meus orientadores, Felipe Maia Galvão França, Diego Dutra e Tiago Alves pelos valiosos conhecimentos recebidos e pela orientação dedicada e profissional que foi fundamental para o amadurecimento e finalização deste trabalho.

Aos meus pais, José Luiz de Aquino Viana e Maria Cristina Faria de Aquino Viana, e aos meus filhos, Luiz Felipe Desouzart de Aquino Viana e Maria Júlia Desouzart de Aquino Viana, pelo apoio e constante incentivo recebidos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D. Sc.)

HORUS: Servidor de Imagens com Suporte as Unidades CSD

Luiz Márcio Faria de Aquino Viana, M. Sc.

Novembro/2022

Orientadores: Felipe França Galvão
Diego Dutra
Tiago Alves.

Programa: Engenharia de Sistemas e Computação

O objetivo deste trabalho e verificar o desempenho das novas unidades de armazenamento com capacidade de processamento embutido, *computational storage drives* (CSD).

Estas unidades de armazenamento realizam o processando embutido dos dados, *in-situ processing*, reduzindo o volume de dados trafegados entre a unidade de armazenamento e a memoria principal e liberando a CPU para outras atividades.

Aplicações que exploram os recursos de processamento das unidades de armazenamento CSD obtém melhor desempenho do sistema e melhoram a eficiência energética no processamento de dados.

Este trabalho apresenta um estudo sobre a otimização das aplicações de processamento de imagens usando unidades de armazenamento com capacidade de processamento (CSD), oferecendo suporte para armazenamento eficiente das imagens geradas por satélites, aerolevantamento, *drones* ou câmeras de vigilância, comparando o desempenho das unidades de armazenamento CSD com soluções convencionais que usam unidades de disco do tipo *solid state disks* (SSD) na busca e no processamento de grandes volumes de dados.

O experimento mostrou que usando unidades de armazenamento com capacidade de processamento embutido, podemos obter um ganhos em *speedup* de 1,69 na conversão de formatos de imagens, de 1,12 na reprojeção das imagens para outro sistema de coordenadas e projeções, e de 1,48 na simplificação e redução do numero de pontos por polegada das imagens, quando comparados com o sistema de referencia que efetua processamento sequencial.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for degree of Doctor of Science (D. Sc.)

HORUS: Image Server with Support for CSD Units

Luiz Márcio Faria de Aquino Viana, M. Sc.

November/2022

Advisors: Felipe França Galvão
Diego Dutra
Tiago Alves.

Department: Engenharia de Sistemas e Computação

The main goal of this work is verify the performance of the new computational storage drives (CSD) with embedded processing support.

This kind of storage drives has support for embedded data processing, named in-situ processing or in-storage compute, which reduce the data volume send and received between the storage unit and the main memory, leaving the CPU free for others activities.

Applications which explore the processing resources of CSD units, get a better system overall performance and better energy saving in the data processing.

This work present a study about imaging processing optimization, using storage drives with in-situ processing (CSD), offering support to store efficiently images generated by satellites, aerial photography, unmanned aerial vehicles (UAVs), or vigilance cameras, and comparing the performance of the CSD units with conventional storage solutions using solid state disks (SSD) in the search and processing of great data volume.

The experiments show which using storage units with in-situ processing, we could get a speedup of 1.69 for image format conversions, 1.12 for image re-projection to different coordinate systems and projections, and 1.48 for image simplification and reduction in the number of points per inches, when compared with a base system with sequential processing.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O crescente volume de dados produzidos pelas redes de sensores e equipamentos de gravação de imagens, torna difícil o processo de captura, transmissão, classificação e identificação dos dados coletados remotamente (Figura 1.1) [1, 2 e 3].

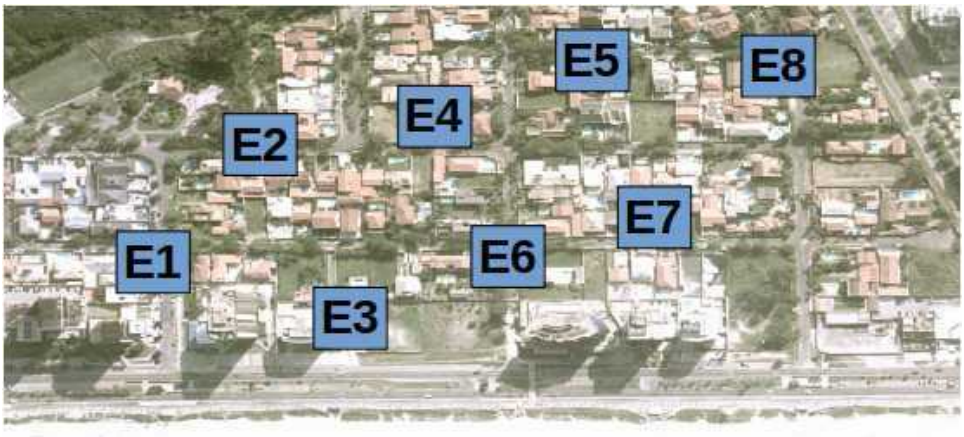

Figura 1.1: Exemplo de dedes de sensores com câmeras espalhadas pela cidade.

Para lidar com o grande volume de informação gerado pelas redes de sensores, é necessário maior espaço de armazenamento e capacidade de processamento da informação [8, 9 e 10].

O uso de múltiplas unidades de armazenamento com capacidade de computação (CSD), permite a distribuição do processamento e do armazenamento das imagens entre as unidades, obtendo maior desempenho no armazenamento e processamento [12 e 13].

As unidades de armazenamento CSD com recursos de processamento *in-situ* fornecem excelente capacidade de otimização das soluções de armazenamento e processamento de imagens.

Desta forma, oferecem a infraestrutura de armazenamento, classificação, indexação e processamento ideal para sistemas de vigilância por câmeras, sistemas de monitoramento por satélites, e sistemas de monitoramento por *unmanned aereal vehicles* (UAVs) [16, 17 e 18].

1.1 MOTIVAÇÕES E OBJETIVOS

O grande volume de informação composta por múltiplos arquivos de dados de grandes dimensões (>400MB) gerados por sistemas de monitoramento por satélites, aerolevantamento e drones, que precisam ser frequentemente processados para conversão entre diferentes formatos de imagens e sistemas de coordenadas e projeção, são ideais para obter proveito da capacidade de processamento fornecida pelas unidades CSD.

Por exemplo, nestes sistemas é muito comum a conversão das imagens armazenadas no formato GeoTIFF para o formato JPEG, permitindo a visualização através do Navegador, e também a conversão entre o sistema de coordenadas e projeção Geográfica - WGS 1984, usado pelo sistema GPS, para UTM – SAD 1989, usado em Projetos de Engenharia [19, 20 e 21].

Além disso, os algorítimos de Inteligencia Computacional consomem muito tempo de processamento no aprendizado e melhoria do modelo, e o ideal e executá-los em *background*, tornando-os independentes da CPU [23, 24 e 25].

Usando unidades CSD com processamento *in-situ*, podemos gerar diferentes modelos de aprendizado, permitindo a classificação e armazenamento dos elementos em um sistema de banco de dados distribuído pelas unidades CSD e orquestrado pela CPU.

O conceito de *federated learning* (FL) permite que as unidades CSD compartilhem a elaboração do modelo de aprendizado entre elas, minimizando a transferência das imagens entre as unidades durante o treinamento do modelo [36 e 37].

Outro recurso necessário aos servidores de imagens, e que pode obter ganhos significativos com o uso de unidades CSD, e a criação de *cache* de imagens com diferentes níveis de precisão, evitando a conversão entre formatos, e permitindo a visualização rápida das imagens em escala reduzida [49].

A utilização da capacidade de processamento das unidades de monitoramento, *edge computing*, permite que sejam realizadas pré-classificações dos dados coletados por cada dispositivo, efetuando um filtro dos dados coletados e proporcionando uma redução em até 20x no volume de dados enviados pelos sensores [38].

As Figuras 1.2 e 1.3, apresentam dois modelos de microcontroladores que oferecem excelente capacidade de processamento, e que podem ser usados para filtrar os dados coletados por cada dispositivo.

Figura 1.2: Raspberry Pi 4 com Processador Broadcom BCM2711.
Quad-core Cortex-A72 (ARM v8) 64bit SoC; Clock 1.5 Ghz; Memória RAM: 4GB DDR4.

Figura 1.3: Jetson Nano com NVIDIA Maxwell e 128 NVIDIA CUDA cores.
Quad-core ARM Cortex-A57 MPCore; 4 GB 64bit LPDDR4; 16GB eMMC 5.1 Flash.

1.2 CONTRIBUIÇÕES DA PESQUISA

Este trabalho apresenta contribuições acadêmicas em diferentes áreas do conhecimento: *artificial inteligence* (AI), *internet of things* (IoT), *cloud computing*, *edge computing*, *federated learning* (FL), *computation storage drive* (CSD), e *geographic information systems* (GIS).

As contribuições em *artificial inteligence* (AI) e *internet of things* (IoT) foram obtidas através da otimização dos processos de captura das imagens por sensores com câmeras de vigilância [45 e 46].

As contribuições em *cloud computing* e *edge computing* foram obtidas através da otimização do processo de criação do modelo de aprendizado, usando conceitos de *federated learning* (FL) e atribuindo parte do processamento de aprendizado do modelo e a decisão de classificação dos dados aos microcontroladores na borda [40].

Contribuição em *computational storage drive* (CSD) com a avaliação dos processos de otimização do armazenamento distribuído entre as unidades, otimização da classificação, indexação e processamento eficiente dos dados [14].

A contribuição na área de *Geographyc Information Systems* (GIS), foi obtida com o desenvolvimento do servidor de imagens, HORUS Image Server, que implementa um sistema de armazenamento e classificação de dados distribuído e com grande capacidade de armazenamento e processamento [51].

1.3 FOCO DA PESQUISA

Esta pesquisa procura, (i) melhorar o desempenho na classificação, busca e visualização de um grande volume de imagens, além de (ii) melhorar a eficiência energética do sistema como um todo, usando unidades CSD no processamento das imagens, e liberando a CPU para outras atividades.

i) Melhoria no desempenho do sistema

O uso de múltiplas unidades de armazenamento CSD, com processamento *in-situ*, permite a distribuição do armazenamento e processamento dos dados, melhorando o desempenho total do sistema.

ii) Melhoria na eficiência energética do sistema

As unidades de armazenamento CSD usam processadores que consomem menos energia, e que estão mais próximos aos dados armazenados.

Desta forma, um mesmo trabalho realizado por um sistema composto por uma CPU e uma unidade de armazenamento do tipo *solid state disk* (SSD), tende a consumir mais tempo de processamento e energia que o mesmo trabalho sendo realizado por um sistema composto pela mesma CPU, e sendo auxiliado por múltiplas unidades de armazenamento CSD, que possuem capacidade de processamento *in-situ* [14].

Este trabalho esta dividido da seguinte forma.

O Capitulo 2, apresenta os fundamentos e trabalhos correlatos, e o Capitulo 3, mostra os conceitos básicos sobre unidades CSD.

O Capitulo 4, apresenta os conceitos sobre processamento de imagens *rasters*, e o Capitulo 5, avalia os recursos de Inteligencia Computacional para analise de imagens.

No Capitulo 6 apresentamos os resultados obtidos nos experimentos, e no Capitulo 7, apresentamos a conclusão e trabalhos futuros.

Apresentamos o ambiente do servidor de imagens, HORUS Image Server, no Apêndice A, e as Bibliografia estudadas no final do trabalho.

Capítulo 2

Fundamentos e Trabalhos Correlatos

Neste capítulo, apresentamos os fundamentos e trabalhos correlatos que formam a base do estudo necessário para elaboração deste trabalho.

Na Seção 2.1, apresentamos os conceitos sobre as unidades de armazenamento do tipo *solid state disk* (SSD), mostrando seus componentes. (a) As unidades de memorias NAND, (b) a unidade de controle, (c) as interfaces de controle, (d) os canais de comunicação, (e) as tabelas de controle *flash translation layer* (FTL), e (f) os processos de otimização implementados nas unidades SSD [52 e 53].

A Seção 2.2, mostra a Especificação NVM-e, que define um protocolo de comunicação entre o servidor e as unidades de armazenamento SSD. Nesta seção, sera apresentado (a) o modelo de transporte com base na memoria, (b) o modelo de transporte com base na mensagem, (c) o modelo de armazenamento NVM, e (d) o modelo de armazenamento com capacidade estendida NVM-e [59].

Em seguida, na Seção 2.3, mostraremos os conceitos sobre o barramento PCI-e. Apresentando (a) a arquitetura do barramento PCI-e, (b) o interfaceamento elétrico do barramento PCI-e, (c) a temporização do barramento PCI-e, e (d) os comandos e funções de controle do barramento PCI-e [58].

2.1 INTRODUÇÃO AS UNIDADES E ARMAZENAMENTO SSD

As unidades de armazenamento do tipo *solid state disk* (SSD) utilizam vários *chips* de memória programáveis do tipo NAND agrupados em conjuntos, onde os *chips* de um mesmo conjunto são conectados pelo mesmo canal de comunicação. A Figura 2.1, apresenta o diagrama de componentes tipico de uma unidade de armazenamento SSD.

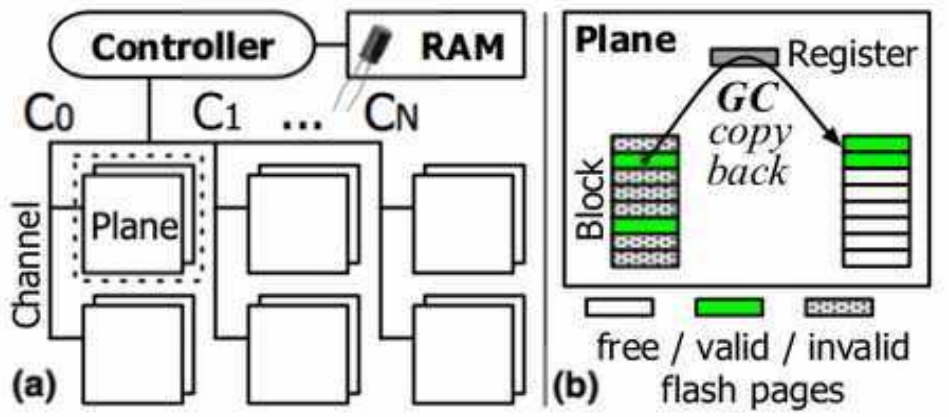

Figura 2.1: Diagrama de componentes tipico da unidade de armazenamento SSD.

Estas unidades possuem uma unidade de controle, responsável por (i) coordenar as operações de leitura, escrita e eliminação dos dados, (ii) coordenar o acesso de leitura e escrita através dos barramentos que interliga os conjuntos de memórias NAND, e (iii) coordenar as operações de compactação e reorganização dos dados, denominada *garbage collection* (GC) [55 e 57].

2.1.1 MEMORIAS NAND

As memórias NAND foram desenvolvidas na década de 1980, e são unidades de memória compostas por varias células de memoria do tipo *flash cell*. As células de memoria *flash cell*, são semelhantes as células de memoria MOSFET usadas nos *chips* de armazenamento *electronically erasable programmable read-only memory* (EEPROM), e são constituídas por dois condutores separados por um elemento dielétrico, como podemos verificar na Figura 2.2.

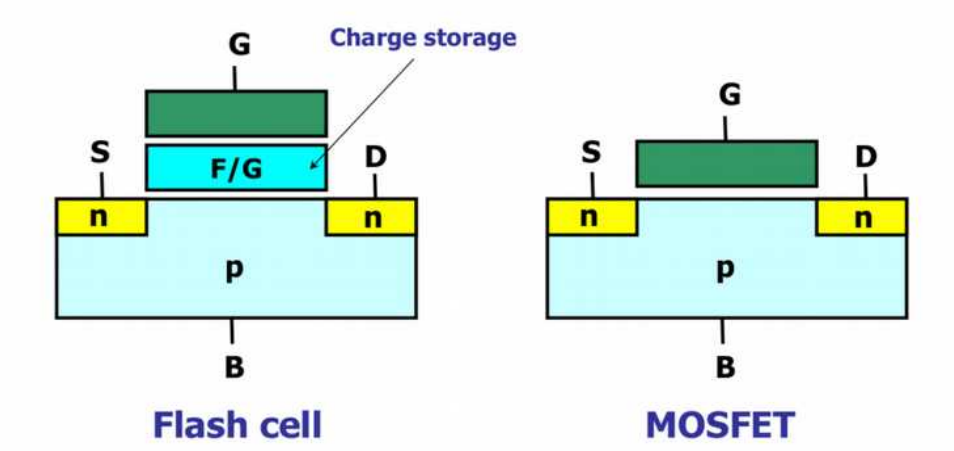

Figura 2.2: Células de memoria *flash cell* e MOSFET.

As células de memória *flash cell*, após o receberem uma carga elétrica (V_G) maior que que um valor limite ($V_G > V_{Threshold}$), são capazes de armazenar a carga elétrica por períodos prolongados de até 10 anos.

As células de memoria *flash cell* possuem 3 portas, que são usadas para leitura, escrita e eliminação dos dados. A Figura 2.3, apresenta a porta do conector de aterramento da célula de memoria ($V_S = 0V$), a porta de ativação do bit de memoria (V_G), que armazena o dado se a tensão aplicada for superior a um valor limite ($V_G > V_{threshold}$), e a porta (V_D), usada para leitura da tensão armazenada, ou para eliminação dos dados quando aplicado uma tensão ($V_D > 0$).

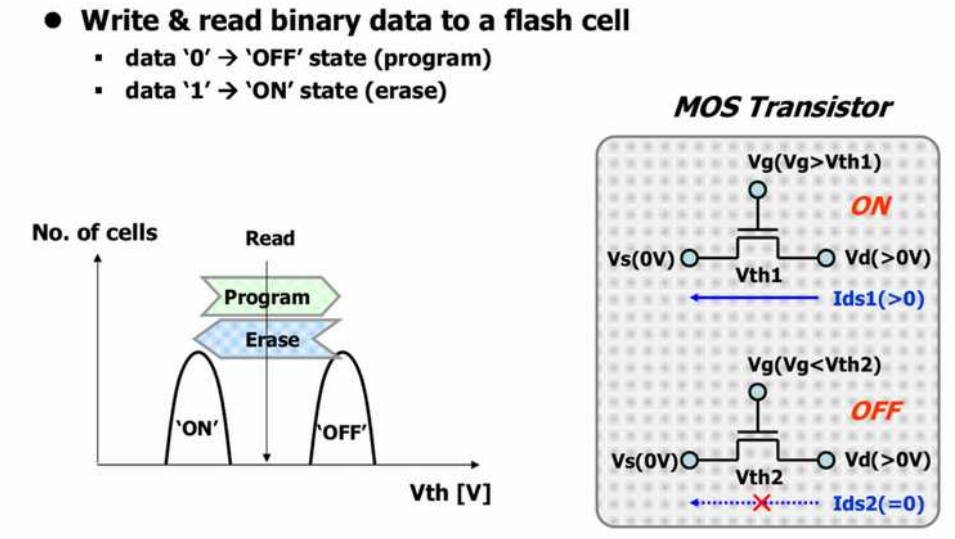

Figura 2.3: *Flash cell* com as portas de leitura, escrita e eliminação dos dados.

Quando uma célula de memória do tipo *flash cell* é carregada eletricamente, a carga elétrica se mantém por um longo período, mas com o decorrer do tempo a carga elétrica sofre atenuações que ao ficar abaixo de um limite impede a identificação do dado armazenado.

A atenuação da carga elétrica ocorre com maior frequência nas células de memoria que realizaram muitas operações de escrita e eliminação dos dados, reduzindo a vida útil das célula de memoria *flash cell [8]*.

2.1.2 UNIDADE DE CONTROLE

Na Figura 2.1, observamos a existência de um *controller*, uma unidade de controle das memorias NAND, responsável por controlar o acesso de leitura, escrita e eliminação dos dados, coordenar o acesso através do barramento, e controlar as operações de compactação e reorganização dos dados, denominada *garbage collection* (GC).

A unidade de controle também e usada para aumentar a vida útil das memorias NAND, através da distribuição equilibrada das operações, sendo responsável por manter todas as células de memória *flash cell* com a mesma frequência de operações de escrita e eliminação de dados *[56]*.

2.1.3 INTERFACES DE CONTROLE

As unidades de armazenamento do tipo *solid state disk* (SSD), suportam interfaces comuns de armazenamento de dados, tais como as interfaces SATA (Serial ATA) e SAS (Serial SCSI).

Porem, devido a elevada taxa de transferência de dados destas unidades de armazenamento, foi elaborado uma nova especificação de interface de acesso as unidades de armazenamento SSD, que usa o padrão de interface PCI Express (PCI-e) e permite a comunicação em velocidades superiores a das interfaces usadas pelas unidades de armazenamento tradicionais [54].

2.1.4 CANAL DE COMUNICAÇÃO

O canal de comunicação, é um barramento de dados que conecta um conjunto de unidades NAND. As unidades de armazenamento SSD atuais, possuem de 4 à 32 canais de comunicação, que são gerenciados pela unidade de controle. Na Figura 4, os canais de comunicação estão identificados por *channel* C_0, C_1, ..., C_N.

Devido ao compartilhamento do mesmo canal de comunicação, pelos *chips* de memoria NAND que fazem parte de um mesmo conjunto, os *chips* de memoria de um mesmo conjunto, efetuam as mesma operação.

Esta organização, aumenta a velocidade das operações, através da distribuição dos dados entre as unidades de memoria NAND de um mesmo conjunto [56].

2.1.5 TABELAS DE CONTROLE *FLASH TRANSLATION LAYER*

As tabelas de controle *flash translation layer* (FTL), que podem ser visualizadas na Figura 2.4, efetuam o mapeamento dos blocos lógicos recebido do sistema operacional, nas páginas e blocos físicos dos *chips* de memoria NAND, e auxiliam a unidade de controle na otimização da vida útil da unidade de armazenamento SSD.

As principais aplicações da *flash translation layer* (FTL) são: (i) o mapeamento dos blocos lógicos em paginas e blocos físicos da unidade de armazenamento SSD, (ii) o mapeamento das áreas com dados inválidos para auxiliar as operações de compactação e reorganização dos dados (GC), e (iii) auxiliar o algorítimo de distribuição dos dados sobre as células de memoria *flash cell* de forma homogênea, com o objetivo de manter equilibrada a frequência de uso de cada célula de memoria, e aumentar a vida útil da unidade de armazenamento SSD [53].

i) Mapeamento dos Blocos Lógicos em Paginas e Blocos Físicos

O processo básico de mapeamento consistem na localização da entrada correspondente a pagina de memoria que contem a referencia ao local de armazenamento do bloco logico na *page mapping table*.

Em seguida, usando a informação da pagina física, na qual esta armazenado os dados referentes ao bloco logico, efetuamos uma pesquisa pela entrada correspondente ao bloco logico na *block mapping table*, e retornamos o bloco físico correspondente.

Na Figura 2.4, a pesquisa pelo bloco logico @5 e realizada diretamente na *page mapping table*. Em seguida, o modulo da divisão do numero do bloco logico, @5, pelo numero de entradas na *block mapping table*, 4 entradas, identifica a entrada da *block mapping table* que contem o numero do bloco físico correspondente, isto e, a pagina 5 e a entrada 1.

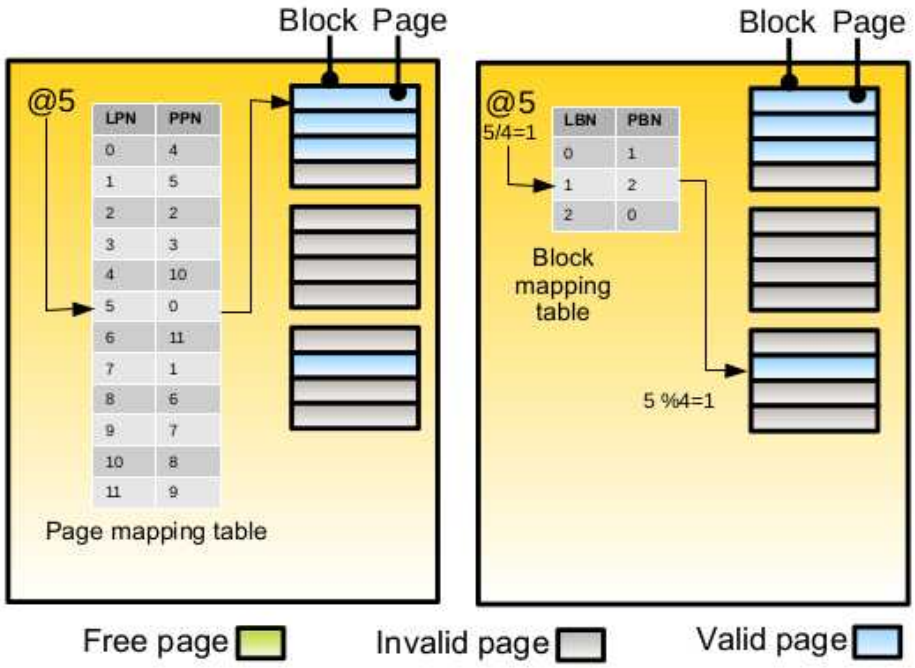

Figura 2.4: Tabelas de controle *flash translation layer* (FTL).

Neste modelo de mapeamento, que usa um endereçamento direto entre o numero do bloco logico e a pagina que contem o bloco físico correspondente, o espaço consumido pela *flash layer table* (FLT), e proporcional a quantidade de blocos físicos existentes na unidade SSD.

Deste modo, uma unidade de armazenamento de 32 GB que foi formatada com blocos físicos de 4 KB, possui 8.388.608 blocos físicos. Se considerarmos que cada entrada da *page mapping table* possui no mínimo 8 bytes, isto e, 2 entradas de endereçamento e 2 *bits* indicadores de pagina livre e pagina valida / invalida, a *page mapping table* ocupara 64 MB de memoria.

A Figura 2.5 apresenta um *chip* de memoria NAND de 64 GB da Toshiba, utilizado em unidades de armazenamento SSD de ate 60 TB que foram projetadas para uso em servidores [60].

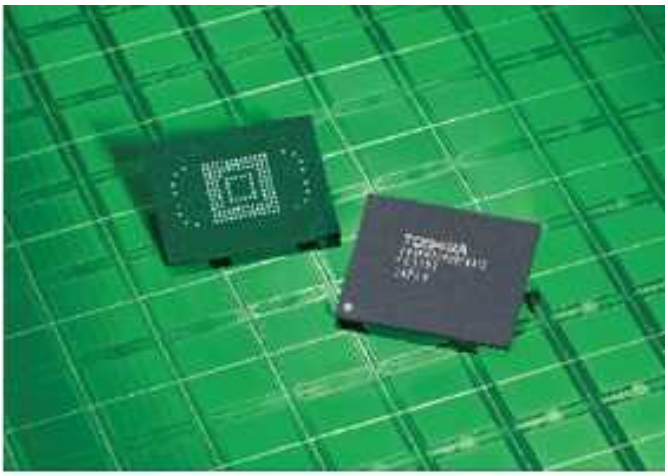

Figura 2.5: *Chip* de memoria NAND de 64 GB da Toshiba.

No segmento de servidores de armazenamento, podemos verificar com facilidade que não e possível aplicar o modelo tradicional de mapeamento direto entre o bloco logico e a entrada correspondente na *page mapping table*, devido ao tamanho ocupado pela *flash layer table* (FLT) das unidades de armazenamento de 60 TB.

A Figura 2.6 apresenta um modelo hibrido de mapeamento, no qual existe uma *global block-based mapping table*, e uma *page mapping table* que contem somente um subconjunto dos dados em memoria, sendo uma alternativa ao modelo de mapeamento apresentado anteriormente, que surgiu com o objetivo de reduzir o tamanho ocupado em memoria pela *flash layer table* (FLT) pelo gerenciador da unidade de armazenamento SSD.

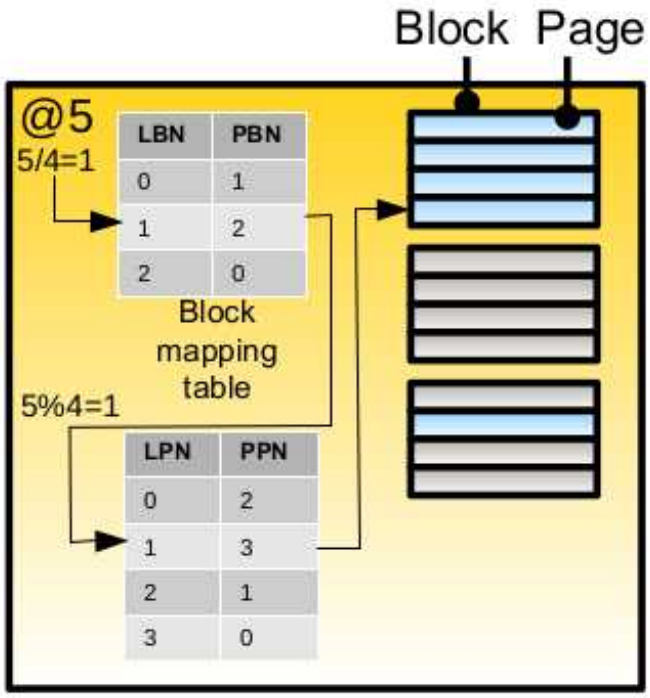

Figura 2.6: Modelo hibrido de tabelas de controle *flash translation layer* (FTL).

Outros modelos de mapeamento foram propostos, com o objetivo de absorver os dados que aguardam a execução do processo de *garbage collection* (GC), para otimização dos espaços livres nas unidades de armazenamento SSD.

ii) Suporte para Operações de Compactação e Reorganização dos Dados (GC)

A Mitsubishi implementou em suas unidades de armazenamento SSD o conceito de *log block FTL*, que armazena em uma fila as opera coes de escrita e leitura que são dependentes, com o objetivo de aguardar o termino da execução do processamento do *garbage collection* (GC) para armazenar os dados.

A Figura 2.7, apresenta a operação realizada pelo processo de *garbage collection* (GC), copiando os dados de um conjunto de blocos físicos para outros, com o objetivo de compactar e reorganizar os dados. Durante esta operação, o processo de *garbage collection* (GC) combina os dados existentes no *log block FTL* e grava os dados na unidade de armazenamento SSD [55].

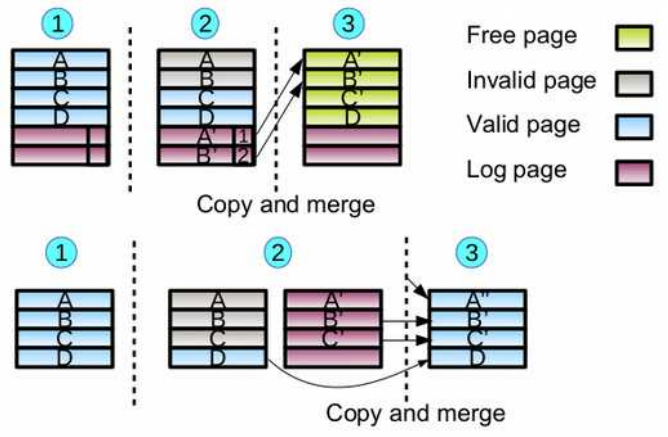

Figura 2.7: Compactação e reorganização efetuada pelo *garbage collection* (GC).

Vários estudos foram realizados com o objetivo de otimizar as operações de *garbage collection* (GC) e a estrutura da *flash layer table* (FLT), com o objetivo de reduzir o espaço ocupado em memoria pela (FLT), otimizar o processo de compactação dos dados, e melhorar a distribuição dos dados pelas células de memoria, *flash cell*, com o objetivo de aumentar a vida útil da unidade de armazenamento SSD.

iii) Algorítimo de Distribuição dos Dados sobre as Células de Memoria *Flash Cell*

Os algorítimos de distribuição dos dados pelas células de memoria *flash cell*, procuram distribuir os dados pelo conjunto de *chips* NAND que estão conectados no mesmo canal de comunicação, mantendo o equilíbrio na frequência de operações de escrita e eliminação de dados realizadas em cada célula de memoria *flash cell*.

Podemos implementar o algorítimo de distribuição dos dados usando 2 critérios de avaliação: (i) usando um algorítimo que avalia a frequência das operações de eliminação de dados, ou (ii) através de um algorítimo que avalia a frequência de operações de escrita que são realizadas em cada bloco de dados.

Alguns mecanismos de distribuição dos dados operam somente sobre os blocos livres e são bastante eficientes em situações na qual existe muito espaço livre disponível na unidade de armazenamento SSD.

Outros algorítimos realizam operações sobre todos os blocos da unidade de armazenamento SSD, com o objetivo de compactar e otimizar os dados armazenados, e prolongar a vida útil do disco [55 e 56].

iv) Algorítimo que Avalia a Frequência das Operações de Eliminação de Dados

Os algorítimos que avaliam a frequência das operações de eliminação de dados realizadas em cada bloco de dados, registram a quantidade de operações realizadas na *block mapping table.*

Periodicamente, efetuam a varredura das entradas da *block mapping table* para verificar a frequência de operações de eliminação de dados. Os blocos de dados com a maior frequência de operações de eliminação de dados são incluídos em uma *hot list*, e os blocos de dados com a menor frequência de operações são incluídos em uma *cold list*.

Para equilibrar a frequência de operações de eliminação de dados realizadas em cada bloco de dados, os blocos de dados que foram incluídos na *hot list*, e com a maior frequência de operações de eliminação de dados, são copiados para blocos livres. Enquanto os blocos de dados com a menor quantidade de operações, e incluídos na *cold list*, são copiados para os blocos de dados com a maior frequência de operações e que ficaram disponíveis.

Esta operação tem com objetivo, transferir os dados estáticos ou que sofrem pouca alteração para os blocos mais usados, reduzindo a frequência de operações de eliminação de dados realizada em cada bloco e aumentar a vida útil da unidade de armazenamento SSD.

v) Algorítimo que Avalia a Frequência das Operações de Escrita dos Dados

Este algorítimo armazena a frequência das operações de eliminação de dados e a frequência de operações de escrita realizadas em cada bloco de dados na *block mapping table*, e periodicamente efetuam a varredura das entradas da *block mapping table* para verificar a frequência de operações de eliminação de dados e de escrita, separando os blocos de dados com a maior frequência de operações em uma *hot list*, e os blocos com a menor frequência de operações em uma *cold list*.

Para equilibrar a frequência de operações de eliminação de dados e escrita, os blocos de dados são intercambiados durante a execução do processo de *garbage collection* (GC) [56].

Neste processo, os blocos de dados incluídos na *hot list*, e com a maior frequência de operações de eliminação de dados e escrita, são copiados para blocos livres, e os blocos de dados com a menor quantidade de operações, incluídos na *cold list*, são copiados para os blocos de dados que ficaram disponíveis.

Esta operação transfere os dados estáticos ou que sofrem pouca alteração para os blocos mais usados, reduzindo a frequência de operações de escrita realizada em cada bloco e aumentar a vida útil da unidade de armazenamento SSD.

2.1.6 PROCESSOS DE OTIMIZAÇÃO

As operações de *garbage collection* (GC) são realizadas com o objetivo compactar e otimizar os dados das unidades de armazenamento SSD, mantendo os blocos de dados de um mesmo arquivo próximos para otimizar as operações de leitura, e equilibrar a frequência de operações de escrita e eliminação de dados realizadas em cada bloco, aumentando a vida útil das unidades de armazenamento SSD.

O execução do processo de *garbage collection* (GC) tem inicio quando a quantidade de espaço livre em disco é inferior a um limite estabelecido, e o canal de comunicação com os conjuntos de memórias NAND fica ocioso por um período de tempo maior que um determinado limite.

No período de inatividade, o processo de *garbage collection* (GC) começa a otimizar os dados gravados na unidade de armazenamento SSD, copiando os dados dos blocos mais usados nas operações de escrita e eliminação de dados, para os blocos livres e com baixa frequência operações.

Durante o processamento do *garbage collection* (GC), este processo pode efetuar a combinação dos dados presentes na *log block FTL*, e em seguida, copiar os dados dos blocos menos usados nas operações de escrita e eliminação de dados, para os blocos que ficaram livres e que possuem maior frequência de operações [57].

A Figura 2.8, apresenta o processo de compactação e otimização dos dados realizado pelo *garbage collection* (GC).

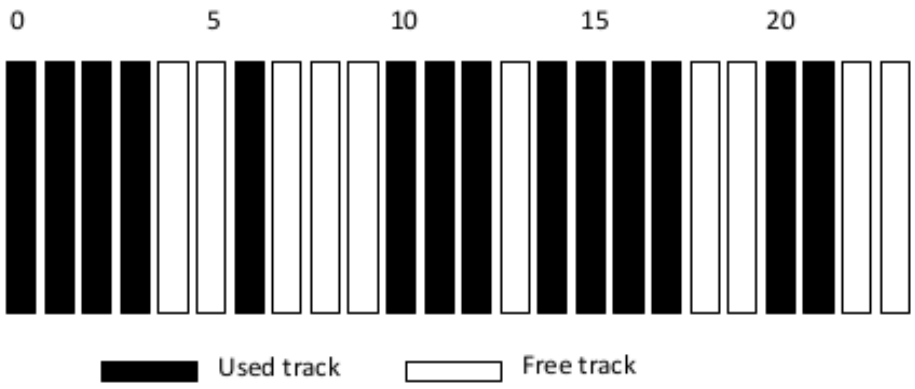

Figura 2.8: Processo de otimização da unidade de armazenamento SSD.

2.2 A ESPECIFICAÇÃO NVM-e

A especificação NVM-e define um protocolo de comunicação entre o servidor hospedeiro e a unidade de armazenamento SSD, padronizando uma série de operações de comunicação de dados e especificando o modelo de transporte, Figura 2.9.

Os modelos de transporte propostos pela especificação NVM-e, são (i) o modelo de acesso direto a memória principal, implementado em arquiteturas de computadores com memoria compartilhada, ou (ii) o modelo de troca de mensagens, valido para os sistemas de memoria compartilhada e para os sistemas de memoria distribuída [59].

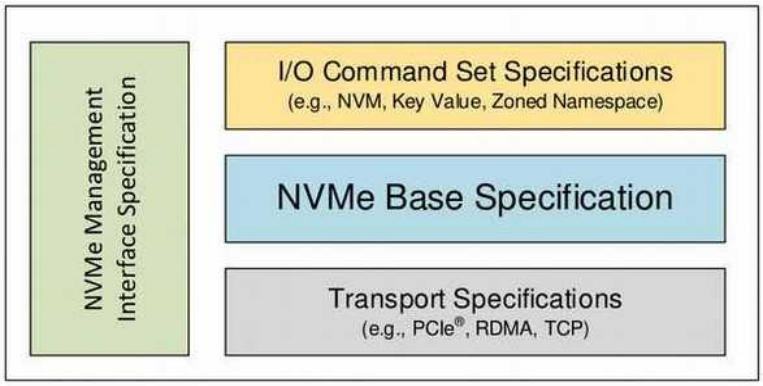

Figura 2.9: Modelo de gerenciamento da interface NVM-e.

2.2.1 MODELO DE TRANSPORTE COM BASE NA MEMORIA

O modelo de transporte com base na memória usa filas de despacho para armazenar as operações enviadas para processamento e que ainda não foram colocadas em execução, e filas de execução para armazenar as operações que estão aguardando o termino do processamento.

As filas de despacho e execução, são alocadas em uma área de memória do servidor hospedeiro, sendo que cada par de filas serão independentes e dedicados a cada núcleo de processamento do computador.

Desta forma, a complexidade gerada pela concorrência no acesso aos dados que estão em um mesmo bloco da unidade de armazenamento SSD e resolvida no nível do sistema operacional, pelo controle da concorrência de leitura e escrita dos arquivos abertos por varias *threads* em diferentes núcleos de processamento.

Este modelo de transporte, que armazena os dados diretamente na memoria principal do computador é mais apropriado para arquiteturas de computadores com memoria compartilhada, onde vários núcleos de processamento compartilham uma mesma memoria principal.

A Figura 2.10, apresenta o dimensionamento das filas de operações do modelo de transporte com base na memoria, em função da arquitetura do sistema e da carga de trabalho esperada.

Sendo tipicamente 1 fila de despacho e 1 fila de execução para cada núcleo de processamento do servidor hospedeiro, evitando a necessidade de *locking* das estruturas de dados, e deixando o controle da concorrência de acesso aos dados no nível de arquivo do sistema operacional [59].

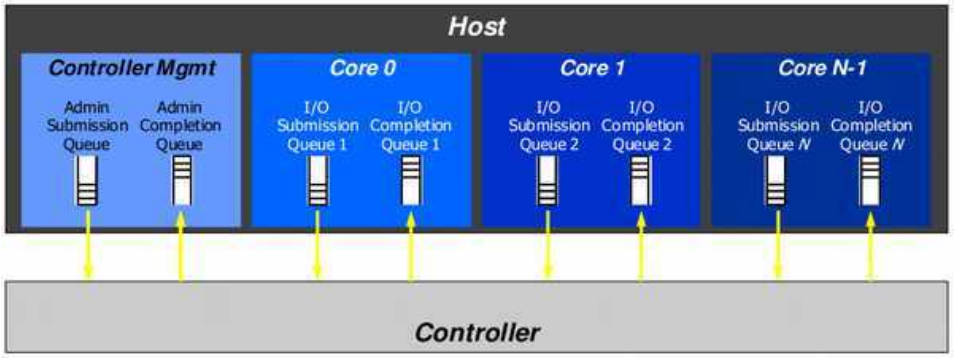

Figura 2.10: Modelo de transporte com base na memoria.

2.2.2 MODELO DE TRANSPORTE COM BASE NA MENSAGEM

O modelo de transporte com base na mensagem, tem como objetivo oferecer suporte para implementação da especificação NVM-e em arquiteturas de computadores com memoria distribuída e que usam um barramento compartilhado pelos processadores.

Neste modelo, assim como ocorre no modelo de transporte com base na memoria, cada núcleo de processamento possui 1 fila de despacho e 1 fila de execução.

Além disso, a unidade de controle do disco SSD também mantém 1 fila de despacho e 1 fila de execução das operações, responsável por sincronizar as operações de acesso a unidade de armazenamento SSD compartilhada, através do mesmo barramento, pelos processadores.

A Figura 2.11, apresenta o conjunto de definições do modelo de transporte com base na mensagem que estão na especificação NVM-e, tais como (i) a definição da arquitetura do modelo NVM-e, (ii) a interface das filas de operações, (iii) os serviços da camada de transporte, e (iv) os modelos de intercomunicação de dados usando PCI-e, Ethernet, InfiniBand, e Fibre Channel [59].

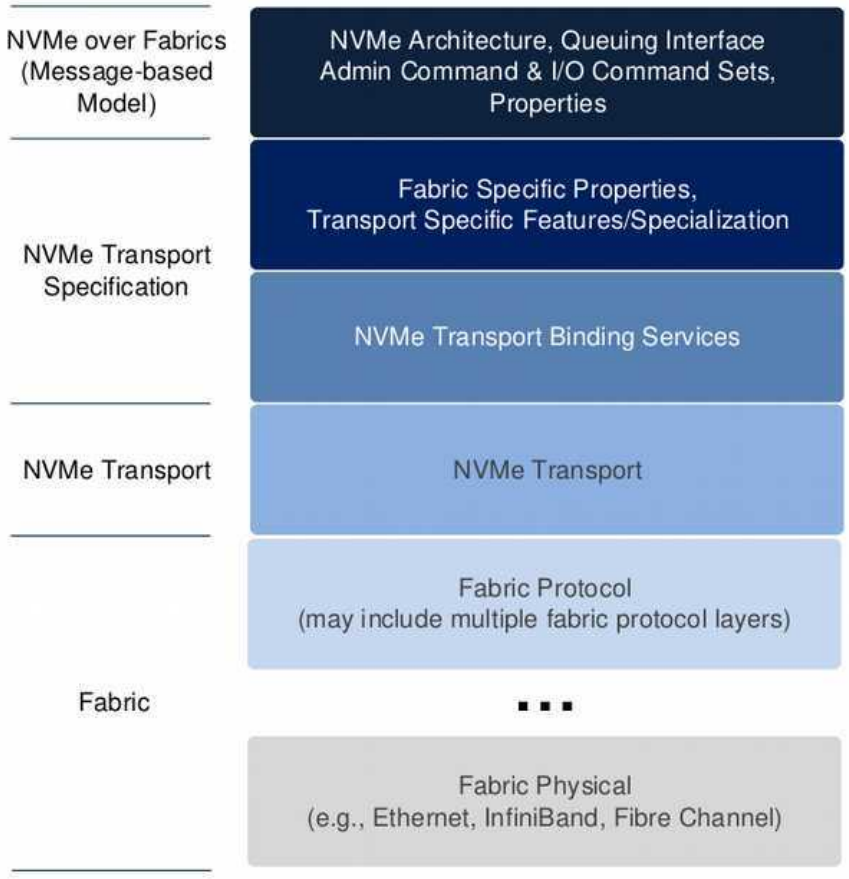

Figura 2.11: Definições do modelo de transporte com base em mensagem.

2.2.3 MODELO DE ARMAZENAMENTO NVM

O modelo de armazenamento NVM, *Non-Volatile Memory*, usa vários conjuntos de *chips* de memorias do tipo *NAND flash* ou *NOR flash*, compostos por células de memorias do tipo *flash cell* ou MOSFET, capazes de manter uma carga elétrica recebida por períodos prolongados de tempo mesmo apos o desligamento dos circuitos.

Estes modelos de armazenamento foram usados inicialmente em pequena escala pelas memorias EEPROM (*Electrically Erasable Programmable Read-Only Memory*), para armazenar o código de inicialização e controle básico dos microcontroladores.

Atualmente, com o avanço da tecnologia de fabricação e da arquitetura dos sistemas de controle das células de memoria, o modelo de armazenamento NVM esta presente todos os equipamentos modernos como automóveis, aviões, geladeiras, televisores, computadores, *smartphones*, *tablets*, e outros.

O modelo de armazenamento NVM que usam *chips* de memoria do tipo *NOR Flash*, oferece excelente desempenho na leitura e escrita dos dados, e permite o acesso direto no nível de *byte*. *Porem*, a organização e controle das células de memoria usadas nestes *chips* ocupam muito espaço, e as memorias do tipo NOR Flash são mais indicadas para armazenamento da inicialização e o controle básico de computadores, *smartphones*, *tablets* e microcontroladores.

As memorias do tipo *NAND flash* organizam os blocos de dados em paginas, que e a menor unidade de leitura e escrita deste tipo de memoria. Esta organização dos dados permite que este tipo de memoria ocupe menos espaço físico e seja utilizada por unidades de armazenamento de grande capacidade.

Atualmente, a Toshiba fornece *chips* de memoria NAND *flash* de ate 64 GB, que são usados por unidades de armazenamento SSD de ate 60 TB.

Porem, para atualizar os dados armazenados em uma memoria do tipo *NAND flash*, e necessário transferir a pagina para uma área de *buffer* na memoria RAM e atualizar os blocos de dados que foram modificados.

Em seguida, os dados contidos na pagina da memoria *NAND flash* são apagados, e somente apos limpar todas as células de memoria contidas na pagina e que os dados atualizados na área de *buffer* são escritos na pagina da memoria *NAND flash [8]*.

2.2.4 MODELO COM CAPACIDADE ESTENDIDA NVM-e

Na Seção 2.2.3, analisamos o modelo de armazenamento NVM, *Non-Volatile Memory*, que usa vários conjuntos de *chips* de memorias do tipo *NAND flash* ou *NOR flash* para armazenamento de dados por longo período de tempo.

O avanço da tecnologia de fabricação e da arquitetura dos sistemas de controle das células dos *chips* de memoria do tipo *NAND flash*, usados por unidades de armazenamento de grande capacidade, trouce novos desafios para o segmento de armazenamento de dados, cujos modelos anteriores de barramento de intercomunicação nao sao mais capazes de fornecer o desempenho requeridos por estes equipamentos.

Deste modo, um grupo de empresas composto por Cisco, Dell, EMC, Intel, Microsoft, Oracle, Samsung, SanDisk e Seagate, se uniram para elaborar a especificação NVM-e que define o protocolo de comunicação, as operações de comunicação de dados, as camadas de transporte de dados, e as interfaces de comunicação suportados [59].

A especificação NVM-e (*Non-Volatile Memory Express*), define um modelo de protocolo otimizado para comunicação direta entre a memória de um servidor hospedeiro com uma variedade de soluções de Intercomunicação, incluindo PCI Express (PCI-e), Ethernet, InfiniBand e Fibre Channel. A Figura 2.12, apresenta os modelos de transporte suportados e os correspondentes canais de intercomunicação suportados.

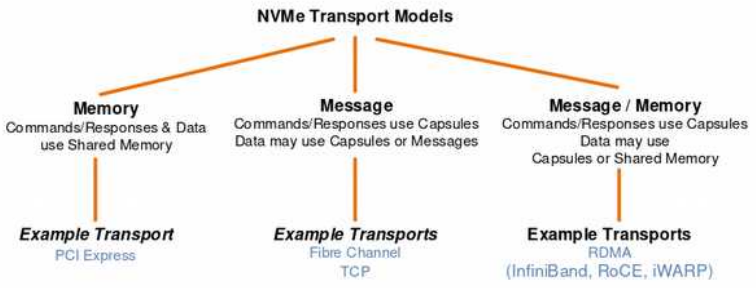

Figura 2.12: Modelos de transporte suportados pela especificação NVM-e.

2.3 CONCEITOS SOBRE O BARRAMENTO PCI-e

O barramento PCI-e, *Peripherical Componente Interconnect Express*, foi desenvolvido na década de 1990, e atualmente e bastante adotado pela industria como um barramento de alta velocidade que opera de forma independente do processador.

Este tipo de barramento aceita até 256 dispositivos como unidades de armazenamento SSD, placas de video, placas de rede, unidades de coprocessamento, e outros.

O barramento PCI-e também pode ser organizado de forma hierarquica permitindo a interconexão com ate 256 barramentos usando *PCI-PCI Bridges* e *PCI Switch*, o que amplia muito a capacidade de comunicação [60].

A Figura 2.13, apresenta a organização do barramento PCI-e com a estrutura hierárquica que atravessa uma porta *PCI Switch*, permitindo a comunicação com outros dispositivos ou barramentos derivados.

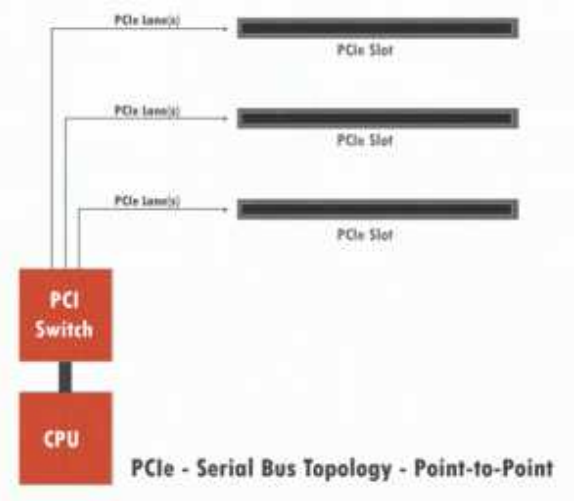

Figura 2.13: Organização do barramento PCI-e.

Todas as operações são realizadas em fluxos contínuos de dados, o que leva o barramento a taxas de transferência superiores a 16 GB/s, e fornece uma solução padronizada para transferência de dados que é independente da placa conectada [60].

A Tabela 2.1, apresenta a comparação entre diferentes especificações da arquitetura do barramento PCI-e.

PCI Architecture	Raw Bit Rate(GT/s)	Interconnect Bandwidth(GB/s)	Bandwidth Lane Direction	Total Bandwidth for x16 link(GB/s)
PCIe 1.1	2.5	2	~ 250 MB/s	~ 8
PCIe 2.0	5	4	~ 500 MB/s	~ 16
PCIe 3.0	8	8	~ 1 GB/s	~ 32
PCIe 4.0	16	16	~ 2 GB/s	~ 16

Tabela 2.1: Comparação entre as diferentes especificações do barramento PCI-e.

2.3.1 ARQUITETURA DO BARRAMENTO PCI-e

A Figura 2.14, apresenta o barramento PCI-e de um computador convencional, onde observamos a existência de três dispositivos de interconexão denominados, *North Bridge*, *South Bridge* e *PCI Bridge*.

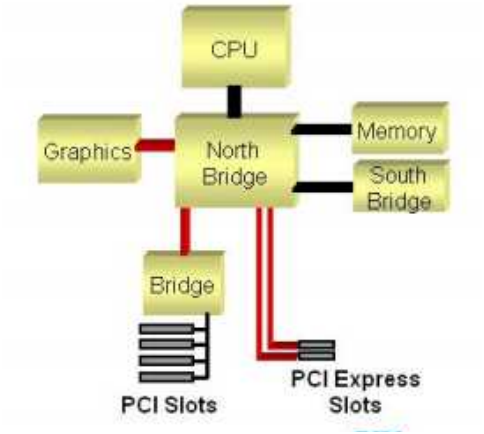

Figura 2.14: Implementação da arquitetura PCI-e em um computador PC.

O dispositivo de interconexão *North Bridge* conecta a CPU, a placa gráfica, a memoria principal e os conectores PCI-e do tipo x1, x4, x8 e x16 existentes nas placas mães dos computadores atuais, Figura 2.15 [61].

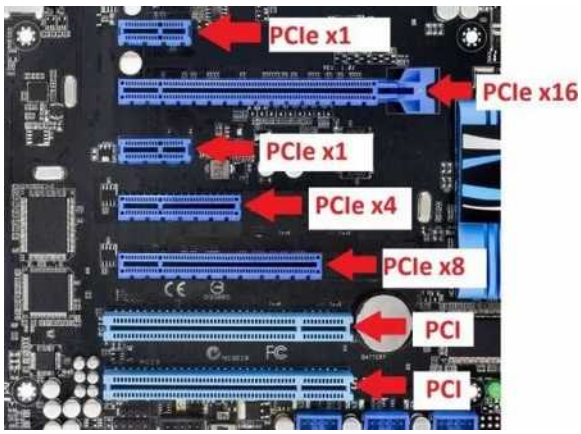

Figura 2.15: Conectores PCI e PCI-e dos tipos x1, x4, x8 e x16 na placa mãe.

A *North Bridge* também conecta os canais de comunicação com a *South Bridge* e a *PCI Bridge*. A *South Bridge* e responsável pela conexão com os dispositivos de entrada e saída padrão como as unidades de armazenamento convencionais e placas de rede. A *PCI Bridge* permite a conexão entre a *North Bridge* com o barramento PCI para conexão de dispositivos legados como as placas SCSI usadas por equipamentos de backup.

2.3.2 INTERFACEAMENTO ELÉTRICO DO BARRAMENTO PCI-e

A Figura 2.16, mostra a pinagem de um *slot* PCI-e, e a Tabela 2.2, apresenta os principais sinais de controle usados neste barramento [60 e 61].

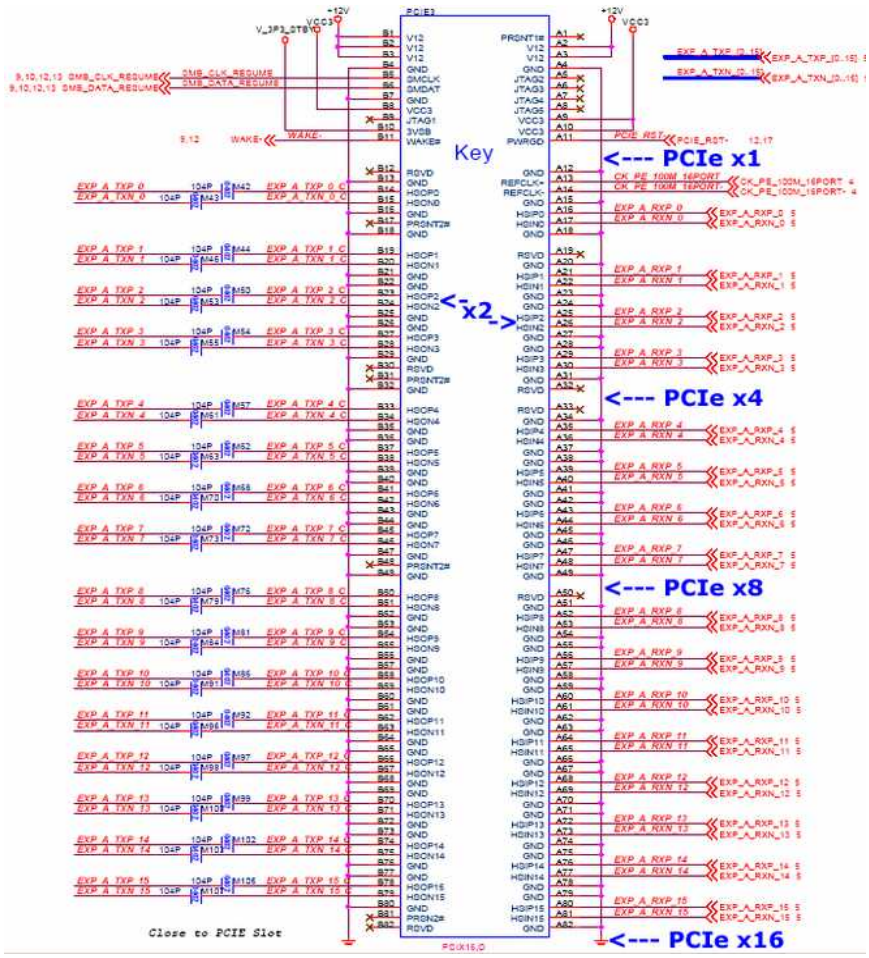

Figura 2.16: Pinagem do Barramento PCI-e x1, x4, x8 e x16.

SMCLK	Sinal de controle que coordena as transações
SMDAT	Sinal de controle que inicia a fase de dados
REFCLK+ / REFCLK-	Sinal do relógio de referencia
HSOP (0-15) / HSON (0-15)	Barramento de saída multiplexado de dados e endereço
HSIP (0-15) / HSIN (0-15)	Barramento de entrada multiplexado de dados e endereço

Tabela 2.2: Sinais de controle do barramento PCI-e.

2.3.3 TEMPORIZAÇÃO DO BARRAMENTO PCI-e

Nesta seção, veremos a temporização do barramento PCI / PCI-e aplicado a uma operação de leitura / escrita de dados em um dispositivo alvo. A Tabela 2.3, lista os elementos que participaram da transação, e as Figuras 2.17 e 2.18, mostram a temporização dos sinais em uma transação de leitura / escrita de dados através do barramento PCI / PCI-e [60 e 61].

Elementos de uma Transacao PCI / PCI-e	
Iniciador	Dispositivo responsável por gerar os sinais de inicialização de uma transação PCI-e.
Alvo	Dispositivo que recebe os sinais do inicializador e participa da transação, enviando e recebendo dados.
Ponte PCI-PCI	Circuito responsável por trocar informações entre dois barramentos PCI-e.
Fase de Endereçamento	As operações de transferência do barramento PCI-e iniciam com uma fase de endereço que demora 1 ciclo de relógio, onde identifica o dispositivo que irá participar da transação e a operação que será realizada. Em seguida, inicia a fase de transferência dos dados.
Transferência de Dados	As transferência dos dados ocorrem em várias fases, após a fase de endereçamento. Operações de escrita no barramento PCI-e transferem os dados do iniciador para o dispositivo. Enquanto, as operações de leitura no barramento PCI-e transferem os dados do dispositivo para o iniciador.

Tabela 2.3: Elementos de uma transação PCI / PCI-e.

No barramento PCI / PCI-e, o sinal PCLK efetua a marcação do relógio e sincroniza todas as transações do barramento.

Para a realização da transação de leitura de dados, Figura 2.17, o endereço do dispositivo alvo deve estar previamente disponível nos terminais HSIP(0-15) e HSIN(0-15), e em seguida, o sinal PFRAME e ativado para indicar a todos os dispositivos conectados ao barramento que o endereço do dispositivo alvo esta sendo anunciado.

Apos 6 ciclos de relógio, o sinal PFRAME e desativado para indicar o termino da fase de identificação do dispositivo alvo e que ira ter inicio a fase de transferência dos dados. O inicio da fase de transferência dos dados e caracterizado pela ativação do sinal PIRDY logo apos a desativação do sinal PFRAME.

Porem, para que ocorra a transferência dos dados, e necessário que durante a fase de endereçamento o dispositivo alvo informe que recebeu a solicitação de inicio da transação, ativando o sinal PDEVSEL, e que em seguida, ative o sinal PTRDY para informar que esta pronto para enviar ou receber os dados.

Apos o termino da transferência dos dados o sinal PIRDY e desativado, e os sinais PDEVSEL e PTRDY são desativados pelo dispositivo alvo [60 e 61].

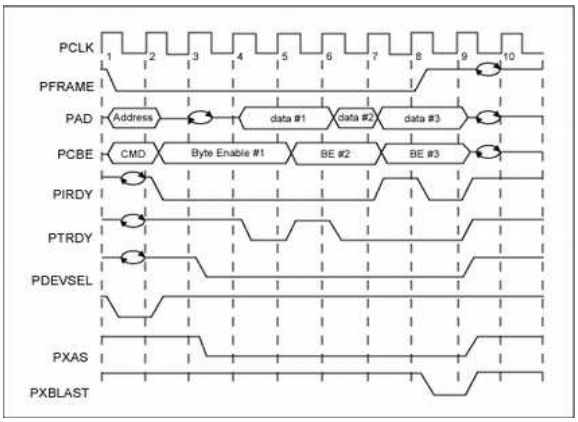

Figura 2.17: Temporização da leitura de dados pelo barramento PCI / PCI-e.

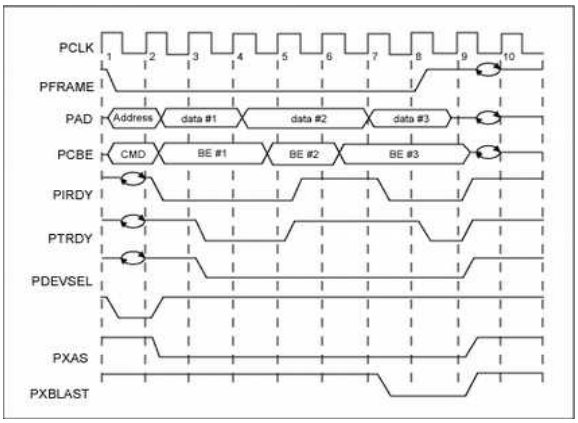

Figura 2.18: Temporização da escrita de dados pelo barramento PCI / PCI-e.

2.3.4 COMANDOS E FUNÇÕES DO BARRAMENTO PCI-e

O barramento PCI-e suporta quatro tipos de transações: (i) transações de acesso a memoria, de (ii) entrada e saída de dados do dispositivo alvo, de (iii) configuração do barramento e de (iv) troca de mensagens. As Tabelas 2.4 e 2.5, apresentam a lista de transações e protocolos suportados pelo barramento PCI-e [60 e 61].

Transaction Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

Tabela 2.4: Transações suportadas no PCI-e.

TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IORd
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message Request with Data	MsgD
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk

Tabela 2.5: Lista de protocolos de transações suportados no PCI-e.

Na Figura 2.19, podemos ver uma transações de leitura de dados, que inicia com o envio do sinal MRd e IORd para indicar ao barramento o inicio da transação. Em seguida, são enviados os sinais de endereço para configuração do barramento, e depois, os dados solicitados são transmitidos do dispositivo alvo para o dispositivo iniciador. Na finalização da transmissão dos dados o sinal CpID é enviado, e se ocorreu algum erro durante a transação, o sinal CpI é enviado após o código do erro [60 e 61].

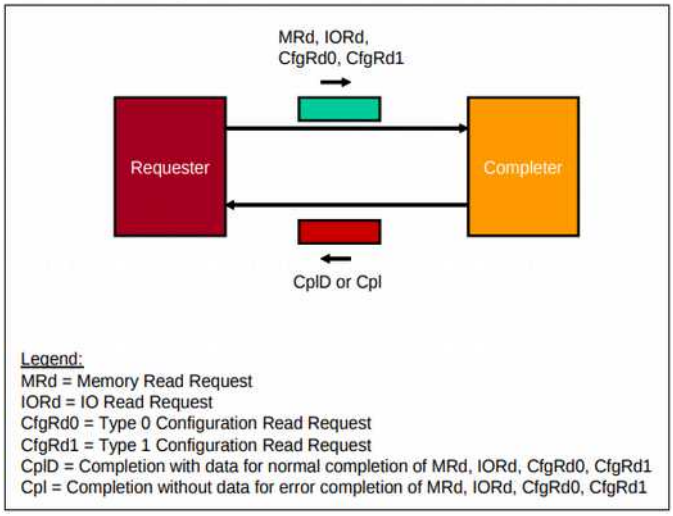

Figura 2.19: Protocolo de leitura de dados através do barramento PCI-e.

Na Figura 2.20, podemos ver uma transações de escrita de dados, que inicia com o envio do sinal IOWr para indicar ao barramento o inicio da transação. Em seguida, são enviados os sinais de endereço para configuração do barramento, e depois, os dados solicitados são transmitidos do dispositivo iniciador para o dispositivo alvo. Na finalização da transmissão dos dados o sinal Cpl é retornado pelo dispositivo alvo [60 e 61].

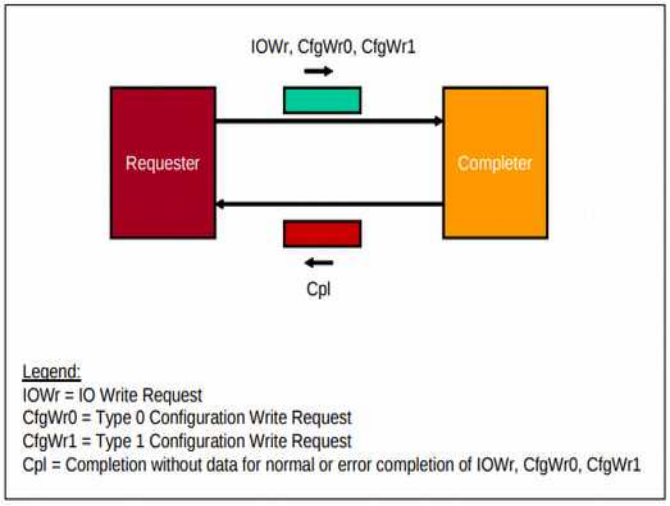

Figura 2.20: Protocolo de escrita de dados através do barramento PCI-e.

Capítulo 3

Conceitos sobre as Unidades CSD

O conceito de *computational storage drive* (CSD) vem sendo amplamente estudado com a aplicação dos recursos de processamento diretamente na unidade de armazenamento, possibilitando que diversos serviços suportados pelo sistema operacional ou pela camada da aplicação estejam disponíveis na unidade de armazenamento, reduzindo o trabalho da CPU e proporcionando maior velocidade na transferência dos dados entre as unidades de armazenamento e memória.

Atualmente o conceito de *computational storage drive* (CSD) esta sendo largamente utilizado em unidades de armazenamento para gravar os dados usando recursos de criptografia e compactação, proporcionando de forma eficiente uma maior segurança e economia de espaço de armazenamento.

Outra aplicação das unidades de armazenamento (CSD), é o gerenciamento de *cache* para otimização de consultas em bancos de dados relacionais, e aplicações de Inteligência Computacional.

Este estudo aplica o conceito de *computational storage drive* (CSD) no aprendizado em Inteligencia Computacional usando Redes Neurais Convolucionais, *convolutional neural network* (CNN), para identificação de texturas, detecção de mudanças e reconhecimento de pessoas e objetos nas imagens [12, 13, 14 e 15].

3.1 ARQUITETURA DAS UNIDADES CSD

Neste estudo avaliamos as unidades de armazenamento de grande capacidade, que usam *chips* NAND, e possuem capacidade de processamento de dados embutido, *in-situ processing*, com o objetivo de otimizar a classificação, busca e processamento de imagens *rasters*.

Foram analisados neste estudo a unidade de armazenamento Newport CSD, fabricada pela empresa NGD Systems. Na Figura 3.1, podemos ver uma imagem da unidade de armazenamento Newport CSD, onde estão destacados o processador ARM Cortex A-53, as Memórias de 16 GB DRAM, e o conjunto de Memórias NAND *Flash*.

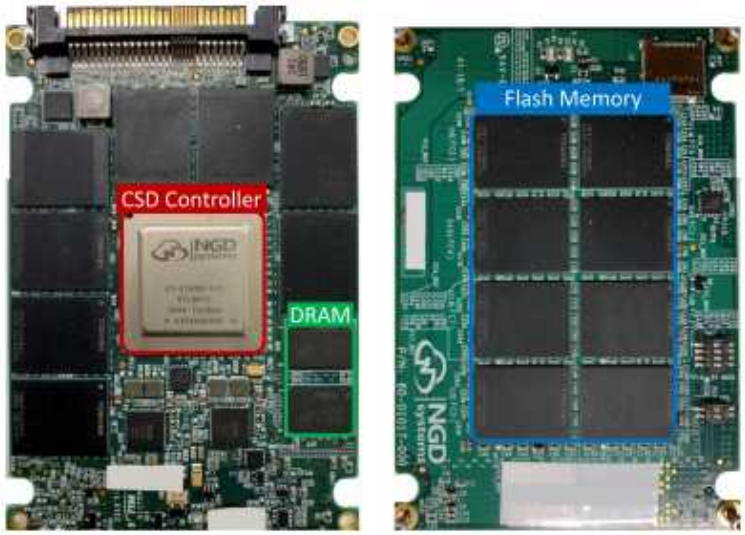

Figura 3.1: Imagem da unidade de armazenamento Newport CSD.

Na Figura 3.2, podemos ver os componentes da unidade de armazenamento Newport CSD, onde agrupamos os componente em três conjuntos com funcionalidades específicas na arquitetura deste sistema. O primeiro conjunto, (i) é formado pelos componentes dedicados a execução das aplicações embarcadas, o segundo, (ii) pelos componentes de comunicação com o barramento PCI-e, e o terceiro, (iii) pelos componentes de controle das memórias NAND *Flash* [60].

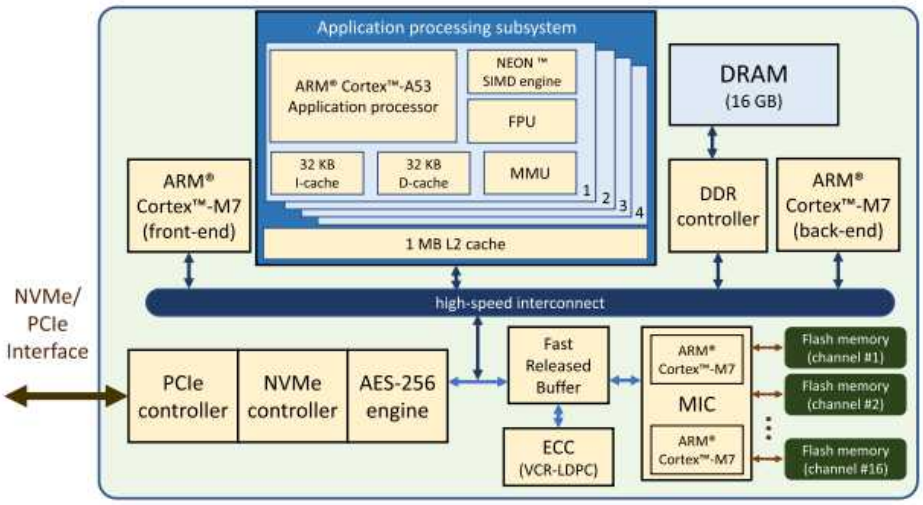

Figura 3.2: Principais componentes da unidade de armazenamento Newport CSD.

i) Componentes Dedicados a Execução das Aplicações Embarcadas

ARM Cortex A-53: Processador da arquitetura ARM Cortex Serie A, de 64 *bits*, com suporte para até 4 núcleos de processamento, com 8 KB I-Cache L1 por núcleo, 8 KB D-Cache L1 por núcleo, suporte opcional para até 2.048 KB Cache L2 para instruções e dados, suporte opcional para instruções SIMD e *floating-point*, e suporte opcional para criptografia.

Na Figura 3.3, apresentamos os principais componentes do processador ARM Cortex A-53, que é usado pela unidade de armazenamento Newport CSD para execução de aplicações de propósito geral, permitindo o processamento de aplicações *in-situ* [63].

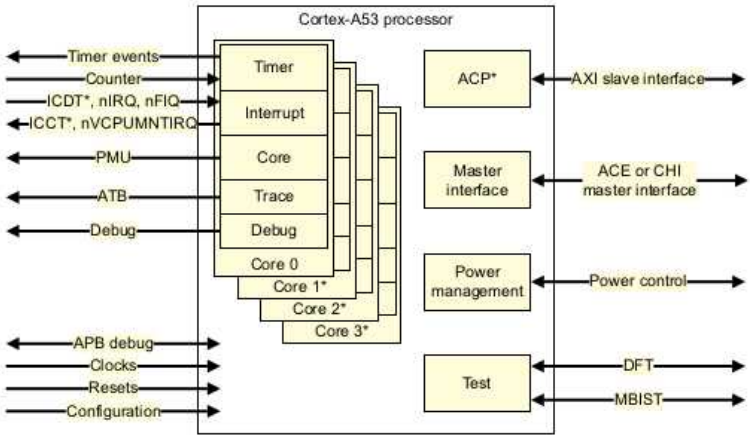

Figura 3.3: Principais componentes do processador ARM Cortex A-53.

Memória de 16 GB DRAM: O conjunto formado pelo processador ARM Cortex A-53 e a memória de 16 GB DRAM, oferecem excelente capacidade de processamento para o desenvolvimento das aplicações que executam na unidade de armazenamento.

ii) Front-End: Componentes de Comunicação com o Barramento PCI-e

ARM Cortex M-7 (*front-end*): Microcontrolador da arquitetura ARM Cortex Serie M, de 32 *bits*, com até 64 KB I-Cache e 64 KB D-Cache L1, com 1 núcleo de processamento, *pipeline superscalar* de 6 estágios, *branch prediction*, até 16 MB de *instruction tightly-coupled memories* (TCM), até 16 MB de *data tightly-coupled memories* (TCM), *hardware divide*, suporte opcional para instruções *floating-point*.

Os sistemas de controle em execução na plataforma ARM Cortex Serie M, residem nas *tightly-coupled memories* (TCM) que podem ter até 16 MB e são independentes para instrução ou dados.

Na Figura 3.4, apresentamos os principais componentes do microcontrolador ARM Cortex M-7, usado no controle do processamento de *front-end* da unidade de armazenamento Newport CSD [64].

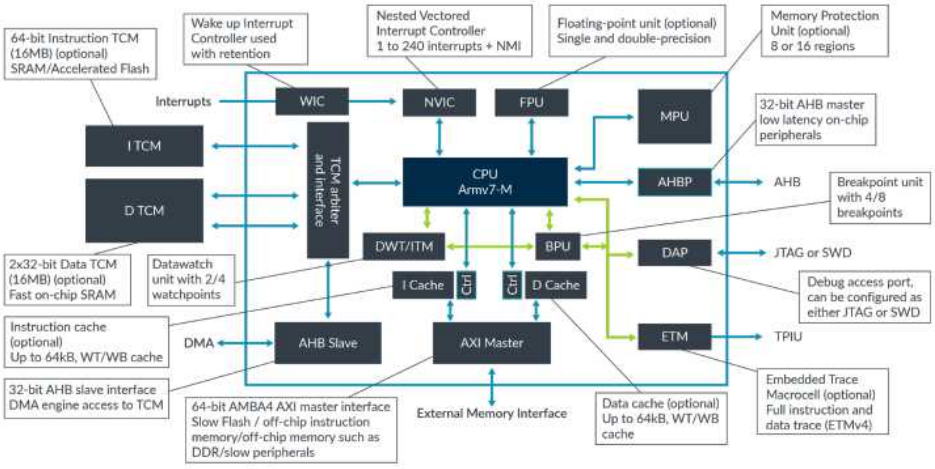

Figura 3.4: Principais componentes do microcontrolador ARM Cortex M-7.

PCI-e / NVM-e Controller: Unidades responsáveis respectivamente pela coordenação e controle do acesso ao barramento PCI-e, e pelo gerenciamento do protocolo de comunicação das transações NVM-e.

AES-256 Engine: Unidade responsável pela segurança e criptografia dos dados enviados e recebidos através do barramento PCI-e.

Fast Released Buffer: *Buffer* de memória de entrada e saída de dados através do barramento PCI-e que fica localizado entre as regiões de *front-end* e *back-end* da unidade de armazenamento Newport CSD.

ECC (VCR – LDPC): Suporte para correção de erros nos dados transportados através do barramento PCI-e, que atua nos dados que passam pelo *buffer* de entrada e saída, reduzindo a quantidade de erros que obrigam a retransmissão de dados através do barramento.

iii) Back-End: Componentes de Controle das Memórias NAND *Flash*

ARM Cortex M-7 (*back-end*): Microcontrolador da arquitetura ARM Cortex Serie M, usado no controle do processamento de *back-end* da unidade de armazenamento Newport CSD [64].

ARM Cortex M-7 (*MIC*): Microcontrolador da arquitetura ARM Cortex Serie M, usado no controle do processamento dos canais de comunicação com as memórias *NAND Flash* [64].

***NAND Flash*:** Memória *NAND Flash* agrupada em conjuntos, onde um microcontrolador ARM Cortex A-7 é responsável exclusivamente pelo gerenciamento de um conjunto de memória *NAND Flash* [65].

Na Figura 3.2, apresentada anteriormente nesta seção, podemos ver que a unidade de armazenamento Newport CSD na imagem, possui 4 conjuntos, com 2 memórias NAND em cada um. Na unidade de armazenamento ilustrada, temos 1 microcontrolador para gerenciamento da comunicação de *front-end*, 1 microcontrolador para gerenciamento da comunicação de back-end, e mais 4 microcontroladores para coordenação da comunicação através dos conjuntos de memórias NAND.

3.2 CAMADA DE SOFTWARE

A camada de software usada pelas unidades de armazenamento Newport CSD estão apoiadas sobre o sistema operacional Linux Ubuntu 16.04.5 LTS [66]. Este sistema oferece os recursos necessários para comunicação entre o servidor hospedeiro e as unidades de armazenamento Newport CSD, organizando a estrutura de armazenamento dos dados de forma distribuída, atuando como um de *cluster* de computadores.

A Figura 3.5, apresenta os componentes da camada de *software* que são tipicamente utilizados pelas aplicações que implementadas nas unidades de armazenamento Newport CSD: Sistema operacional Linux Ubuntu 16.04.5 LTS no servidor hospedeiro e nas unidades de armazenamento Newport CSD [], camada de rede usando o protocolo TCP/IP, túnel para comunicação TCP/IP sobre PCI-e/NVM-e, *containers* para isolamento das aplicações clientes em execução em cada unidade de armazenamento, e aplicações específicas instaladas no servidor hospedeiro e instanciadas em *containers* nas unidades de armazenamento Newport CSD [70].

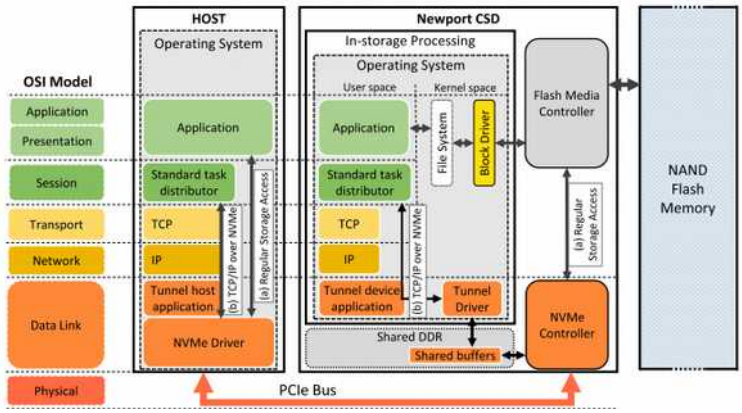

Figura 3.5: Componentes da camada de software do Newport CSD.

3.3 PRINCIPAIS RECURSOS

A Tabela 3.1, apresenta as unidades de armazenamento CSD, fabricadas pela empresa NGD Systems. Podemos observar que as unidades de armazenamento comercializadas atualmente pela empresa tem capacidade de armazenamento de até 32 TB, e alcançam uma taxa de transferência de dados de até 1,6 GB/s para acesso sequencial e de até 200 KB/s para acesso aleatório aos blocos do disco [70].

	M.2 - 2280	M.2 - 22110	EDSFF – E1.S	U.2 – 2.5-inch SFF
Raw Capacity		Up to 8TB	Up to 12TB	Up to 32TB
Read Performance[1]	4K Random – 200K		128K+ Sequential – 1.6GB/s	
Write Performance[1]	4K Random 35K	4K Random 35K	4K Random 200K	4K Random 200K
Avg. Active Power (W)	6	8	12	12
Supply Voltage (V)	3.3	3.3	12	12
Temp Ranges With Airflow[2] (°C)	0 to 60 Additional Temp Ranges Available			
Dimension – W x L x H (mm)	22 x 80 x 5	22 x 110 x 5	31.5 x 111.5 x 6	69.85 x 100.45 x 15

Tabela 3.1: Especificação dos produtos Newport Platform da NGD Systems.

Os produtos possuem um processador ARM Cortex A-53 de 64 bits, com 4 Núcleos de Processamento, e 16 GB de Memória DRAM, que são dedicados a execução de aplicações *in-situ*.

A Tabela 3.2, apresenta a unidade de armazenamento SmartSSD, fabricada pela empresa Samsung. Podemos observar que as unidades de armazenamento comercializadas atualmente pela empresa tem capacidade de armazenamento de 4 TB, e alcança uma taxa de transferência de dados de até 3,3 GB/s para acesso sequencial [71].

A elevada taxa de transferência obtida pelo equipamento SmartSSD da Samsung, é alcançada devido a unidade de compressão de dados que implementa o algorítimo LZ4 e reduz o volume de dados transferidos pelo barramento PCI-e.

Form Factor	2.5" (U.2)	
Storage Capacity	3.84TB	
Host Interface	Single Port PCIe Gen3x4	
Spec Compliance	NVMe spec rev. 1.3, PCIe based specification rev 3.0, NVMe Management Interface (NVMe MI) 1.0	
NAND Flash Memory	Enterprise Class SSD Controller	
	Samsung V-NAND®	
	Write Endurance	1DWPD for 5 Years
	Sequential Read (128KB)	Up to 3,300 MB/sec
	Sequential Write (128KB)	Up to 2,000 MB/sec
	Random Write (4KB)	Up to 800,000 IOPS
	Random Read (4KB)	Up to 110,000 IOPS
	Uncorrectable Bit Error Rate (UBER)	1 sector per 10^17 bits read
	Mean Time Between Failure (MTBF)	2,000,000 hours
Hot Plug Support	Power Loss Protection Included Inrush Current Protection Included	
Dimensions	60 x 100 x 15 mm	
Weight	400 grams	

Tabela 3.2: Especificação do produto Samsung SmartSSD.

O produto Samsung SmartSSD, é uma unidade de armazenamento CSD que utiliza um módulo FPGA, com capacidade de efetuar operações e otimizar o processamento destas unidades

3.4 MODULO FPGA

A unidade de armazenamento Samsung Smart SSD, possui um módulo FPGA que pode ser personalizado de acordo com a aplicação que é destinada ao equipamento. A Tabela 3.4, apresenta os recursos de configuração do módulo FPGA deste equipamento [71].

Programmable Hardware Accelerator (FPGA)	Xilinx Kintex™ Ultrascale+ KU15P FPGA	
	System Logic Cells	1.143 Million
	Available LUTs for acceleration tasks	Approx. 300K
	DSP Slices	1,968
	Internal Distributed RAM	34.6 Mbit
	Internal UltraRAM	36.0 Mbit
	Accelerator-dedicated RAM	4 Gbyte DDR4 SDRAM @2400 Mbps
	Speed Grade	-2LE
Temperature	Operating	0 to 70° C
	Non-Operating	-40 to 85° C
Humidity	Non-Operating	5% to 95%, Non-condensing
Power Consumption	Power Management	Dynamic power management and throttling
	Active	<=25W
	Idle	18W
Vibration	Non-Operating	20G (10~2000HZ, sweep sine)
Shock	Non-Operating	1500G, Duration 0.5 msec, Half-Sine Wave

Tabela 3.3: Especificação do módulo FPGA do Samsung SmartSSD.

Por exemplo, para o servidor de imagens, HORUS Image Server, podemos incluir no módulo FPGA um subconjunto de instruções complexas e frequentemente usadas nos processos de compressão, descompressão, conversão, simplificação, e modificação de sistemas de coordenadas e projeção das imagens *rasters*.

Neste capitulo, apresentamos o conceito de *computational storage drive* (CSD) e avaliamos os equipamentos Newport CSD de fabricação da empresa NGD Systems, e o equipamento Samsung SmartSSD de fabricação da empresa Samsung.

No próximo capítulo, estudaremos os conceitos sobre processamento de imagens *rasters*, analisando os processos de classificação de imagens usando redes neurais, conversão de formatos, conversão de sistema coordenadas e projeção, e simplificação de imagens.

Capítulo 4

Processamento de Imagens

Os sistemas de informações geográficas, *geographic information systems* (GIS), associam elementos vetoriais e imagens *rasters*, que representam áreas de levantamento na superfície terrestre, com tabelas de dados para apresentar as informações associadas na forma de mapas, Figura 4.1.

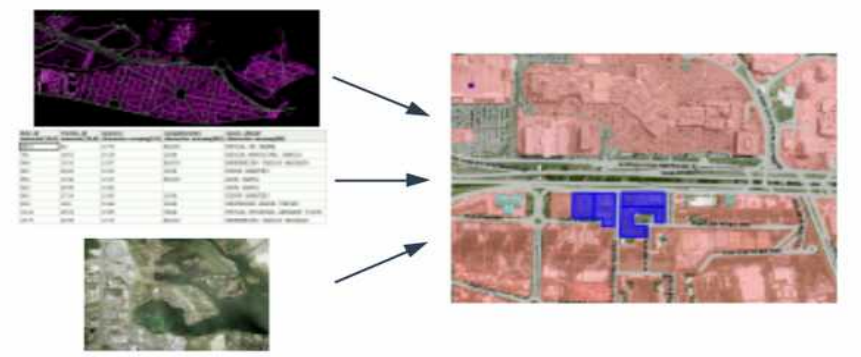

Figura 4.1: Associação de elementos vetoriais, imagens *rasters* e tabelas de dados.

Em um sistema de informações geográficas (GIS) podemos obter informações sobre propriedades, equipamentos e instalações urbanas, além das redes de energia, água e esgoto, dos pontos de abastecimento, e de locais de entrega, e etc. Para implementação de um sistema de informações geográficas preciso, é fundamental a definição de um sistema de coordenadas e de projeção adequados [19, 20, 21 e 22].

4.1 CONDIÇÕES INICIAIS

No inicio deste estudo, contactei os coordenadores das equipes de geoprocessamento da TRANSPETRO, IBGE e EMBRAPA, com o objetivo de verificar se o estudo que estava sendo proposto poderia no futuro ter uma aplicação pratica entre as empresas consultadas.

Em reunião com a equipe de cartografia do IBGE, eles informaram que a realização do Censo Demográfico é sempre um grande desafio para eles, e que para os próximos levantamentos, eles estão avaliando o uso de Inteligência Computacional para processamento das imagens das 480.000 zonas censitárias existentes no Brasil, identificando áreas do território Brasileiro com novas moradias, instalações comerciais e instalações industriais.

Desta forma, o IBGE iniciou um estudo usando imagens de satélites comerciais com precisão de 8 metros para serem usadas em futuros levantamentos. Estas imagens serão comparadas com imagens e elementos vetoriais obtidos em levantamentos de anos anteriores para identificar mudanças nas áreas censitárias.

Cada cena adquirida possui 1 imagem *true color e* 11 imagens em tons de cinza, que representam diferentes faixas de frequência do espectro luminoso, variando de 412 nm à 2.320 nm. O espaço estimado ocupado pelas imagens referentes a somente 1 levantamento de todo o território Brasileiro é de 4.678 GB (5 TB).

Na Tabela 4.1, estão listadas as diferentes faixas de frequencia que foram adquiridas pelo IBGE para avaliar o uso da Inteligência Computacional no processamento das imagens das zonas censitárias.

As faixas de frequência selecionadas pelo IBGE nas imagens em tons de cinza, foram escolhidas pelo interesse deles em identificar com maior facilidade áreas da imagem com maior quantidade de grama natural, concreto, água e etc.

Estas imagens serão usadas para facilitar a identificação dos objetos que possuem maior reflectância nas faixas do espectro luminoso selecionadas, e orientar o algorítimo de Inteligência Computacional no seu aprendizado.

		Frequencia			Tamanho
		Min	Max	Central	Faixa
#	BANDA	(nm)	(nm)	(nm)	(nm)
0	B1	412.0	456.0	442.7	44.0
1	B2	456.0	533.0	492.7	77.0
2	B3	538.0	583.0	559.8	45.0
3	B4	646.0	684.0	664.6	38.0
4	B5	695.0	714.0	704.1	19.0
5	B6	731.0	749.0	740.5	18.0
6	B7	769.0	797.0	782.8	28.0
7	B8	760.0	907.0	832.8	147.0
8	B8A	837.0	881.0	864.7	44.0
9	B9	932.0	958.0	945.1	26.0
10	B10	1337.0	1412.0	1373.5	75.0
11	B11	1539.0	1682.0	1613.7	143.0
12	B12	2078.0	2320.0	2202.4	242.0
13	TCI	400.0	700.0	550.0	300.0

Tabela 4.1: Faixas de frequência do espectro luminoso das imagens do IBGE.

Na Figura 4.2, podemos ver o gráfico de reflectância de alguns objetos reconhecidos em imagens de satélites. Neste gráfico, podemos observar que a grama natural possui maior reflectância na faixa do espectro luminoso de 700 nm à 900 nm (Banda B8 na Tabela 4.1) e que a água possui maior reflectância na faixa de 550 nm à 600 nm (Banda B3 na Tabela 4.1).

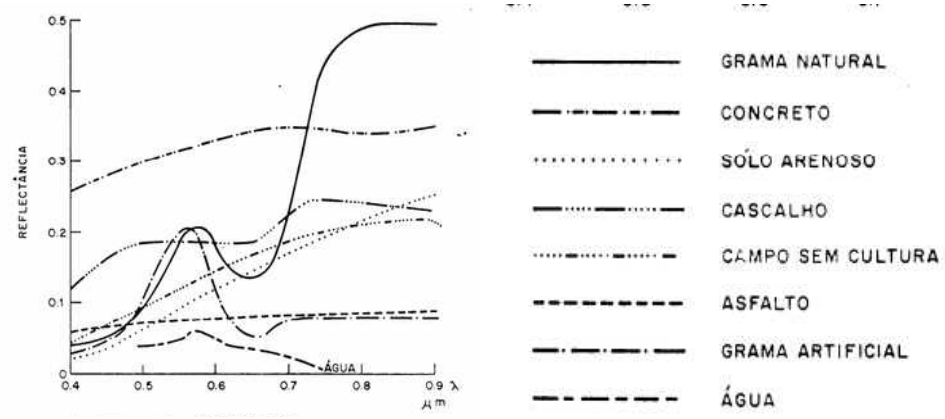

Figura 4.2: Gráfico de reflectância de alguns elementos.

Existem atualmente imagens de satélites com precisão inferior a 50 cm, onde 1 *pixel* representa uma região de 50 cm^2 da área de levantamento. Na Figura 4.3, podemos comparar a qualidade das imagens de uma mesma cena com diferentes níveis de precisão.

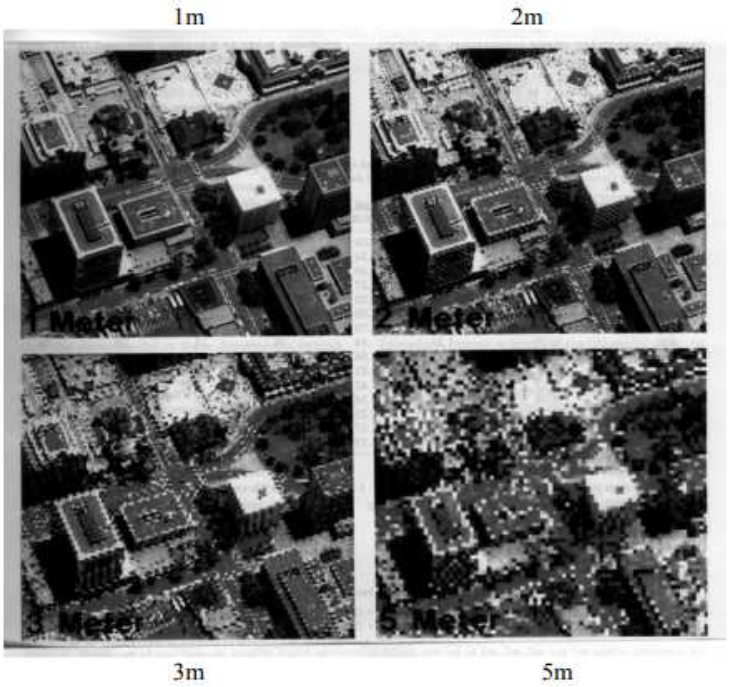

Figura 4.3: Imagens de satélite com diferentes níveis de precisão.

A Figura 4.4, mostra um exemplo de imagem com precisão de 8m adquirida pelo IBGE, que optou por imagens com menos precisão porque para o levantamento censitário não há necessidade de maior nível de detalhamento nas imagens.

O espaço de armazenamento consumido pelas imagens com precisão de 8 m, é inferior ao espaço ocupado por imagens com precisão de 50 cm. Além disso, o processamento de imagens menores é muito mais rápido.

Se o IBGE adquirisse imagens de 50 cm, o espaço necessário para armazenar todo o território Brasileiro seria de 1.280 TB, ou 256 vezes o espaço de 5 TB, necessário para armazenar as imagens com precisão de 8 m que foram adquiridas.

Figura 4.4: Imagem adquirida pelo IBGE com precisão de 8 metros.

O uso de imagens *multispectral* com 12 faixas selecionadas pelo IBGE, auxiliam a identificação de áreas de concentração urbana, áreas de cultivo, áreas de floresta nativa, rios, lagoas, e etc., facilitando a analise das imagens através da sobreposição delas com a imagem *true color* obtida da mesma área de levantamento.

Por exemplo, a sobreposição da imagem *true color* com a imagem em tons de cinza B8, destaca as regiões com maior incidência de grama natural. A sobreposição da imagem *true color* com a imagem em tons de cinza B3, destaca as regiões alagadas, os rios e os lagos da imagem.

Além de usar imagens *multispectral* para identificação dos elementos em função de sua reflectância, o IBGE irá comparar as novas imagens com imagens de levantamentos anteriores e com o cadastro de mapas vetoriais da instituição. Este tipo de comparação irá permitir a identificação de novas áreas urbanas para auxiliar o levantamento de dados para o Censo Demográfico.

4.2 GEORREFERENCIAMENTO DE IMAGENS *RASTERS*

Os equipamentos de sensoriamento remoto e as imagens geradas por satélites, aeronaves, e *drones*, são obtidas em um sistema de coordenadas geográficas sobre a superfície da Terra. Neste sistema, qualquer ponto na superfície da terra é representado pelas coordenadas de latitude e longitude, Figura 4.5 [19, 20, 21 e 22].

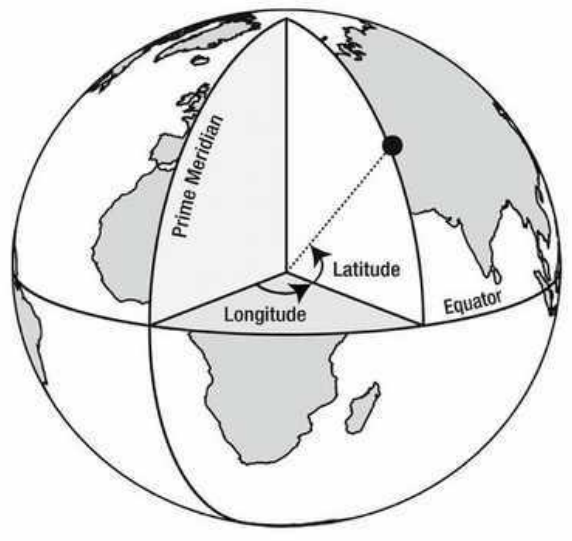

Figura 4.5: Sistema de coordenadas geográficas.

Definição de um Sistemas de Coordenadas

Se o nosso planeta fosse uma esfera perfeita, com raio de curvatura de 6.400 Km de extensão, e escolhêssemos como origem o centro da Terra, poderíamos aplicar diretamente as coordenadas geográficas no georreferenciamento de objetos na superfície.

Porém, a Terra não é uma esfera perfeita, tem diâmetro equatorial maior que o diâmetro nos polos, e possui montanhas e vales. Desta forma, a Terra e todos os planetas rochosos que conhecemos possuem uma forma denominada Geoide, Figura 4.6.

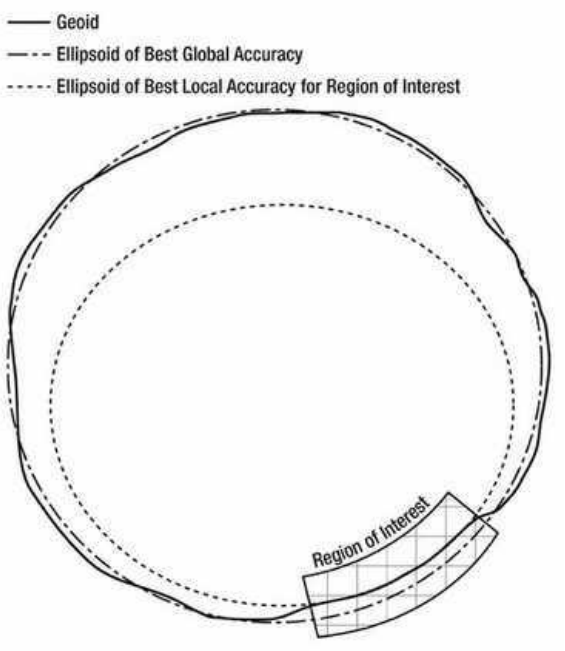

Figura 4.6: Definição do Elipsoide e Datum de referência.

A representação da superfície da Terra por um Geoide não é muito prática, porque não existe uma fórmula matemática direta para localizar precisamente uma pessoa que está no topo do Everest ou em sua base.

Deste modo, procuramos aproximar a superfície da Terra por um Elipsoide que é a forma geométrica que mais se aproxima do Geoide, e para o qual podemos criar uma fórmula matemática que possibilite localizar de forma precisa um objeto em sua superfície.

Entretanto, a Terra é um planeta muito grande, e quando usamos um modelo de Elipsoide único para a sua representação, perdemos precisão devido a forma do planeta e as variações do relevo. Deste modo, definimos diversos Elipsoides que são específicos para cada finalidade e região.

Quando associamos o Elipsoide de referência com a origem e a direção dos vetores de coordenadas, obtemos a definição de um *Datum*. O *Datum* é a origem de qualquer sistema de coordenadas.

No Brasil utilizamos com frequência o *South American Datum* 1969 (SAD69) e o SIRGAS2000, que foram definidos usando um Elipsoide mais ajustado para projetos realizados em território Brasileiro. Nos equipamentos GPS, os dados obtidos nos levantamentos usam WGS 1984 que é uma referência de Elipsoide para uso Global.

Para algumas aplicações, a utilização de coordenadas geográficas não é prática, porque pode ser necessário extrair medidas de comprimento e área do mapa. Para facilitar a análise dos dados estes podem ser projetados em um mapa utilizando diferentes métodos de projeção, Figura 4.7 [19, 20, 21 e 22].

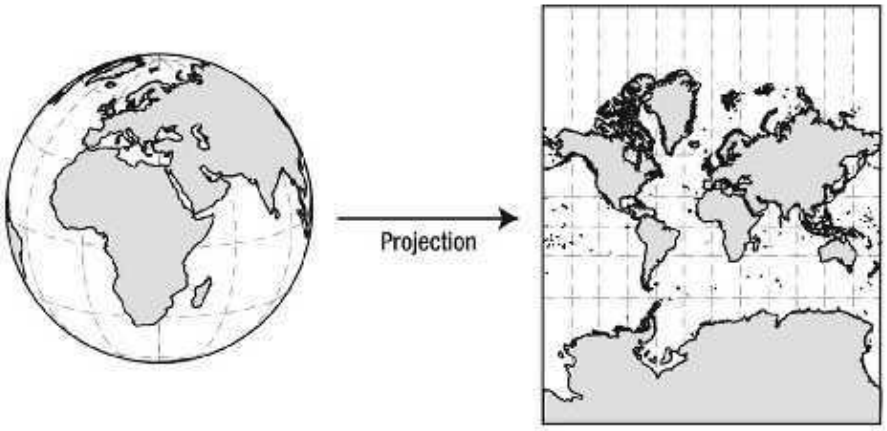

Figura 4.7: Projeção das coordenadas geográficas em um mapa.

Sistemas de Coordenadas Projetadas UTM

A projeção *Universal Transverse Mercator* (UTM) preserva a forma dos objetos e possui melhor precisão porque atua em faixas menores da superfície terrestre.

Na Figura 4.8, podemos ver um mapa do globo terrestre com as 60 zonas definidas pelo sistema de projeção UTM. O sistema de projeção UTM converte os elementos gráficos de coordenada geográfica em latitude e longitude sobre um Elipsoide, para coordenadas métricas sobre os eixos X e Y em um Plano.

Por representar a projeção de um Elipsoide em um Plano, as medidas de comprimento e área no sistema de coordenadas UTM sofrem distorções, sendo que as medidas são mais precisas próximo ao eixo central de cada zona de projeção e menos precisa nas extremidades. Para minimizar os efeitos da distorção, o sistema UTM define 60 zonas de projeção.

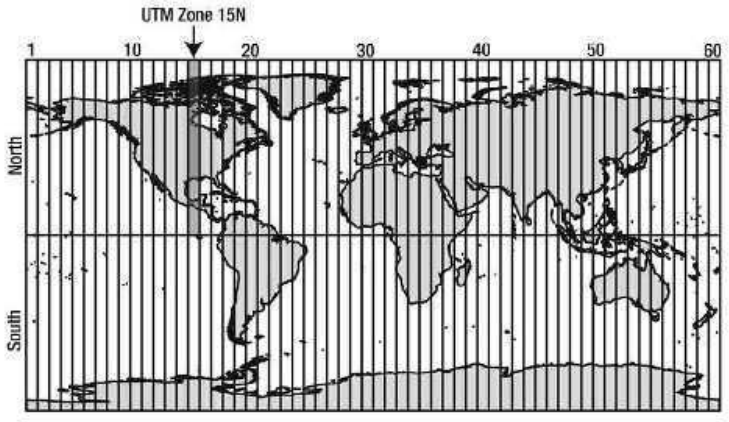

Figura 4.8: Imagens das 60 zonas de projeção do sistema UTM.

O sistema de projeção UTM é muito usado em obras de construção de estradas, pontes, linhas de transmissão e barragens, devido a facilidade de obtenção de medidas aproximadas e comprimento e área.

Este sistema de projeção também é bastante usado na atualização do cadastro de engenharia em empresas de telecomunicação, energia, água e gás natural. Porque a equipe que realiza o levantamento em campo consegue obter medidas de comprimento e área com facilidade [19, 20, 21 e 22].

4.3 PROCESSAMENTO DE IMAGENS *RASTERS*

As imagens *rasters* são formadas por uma matriz de pontos, onde cada ponto pode ser representado por um conjunto de bits de acordo com a profundidade de cores da imagem.

Por exemplo, imagens em preto e branco podem ser representadas como uma matriz de pontos de 1 bit (0=Preto ou 1=Branco). Imagens com 8 cores representam cada ponto da matriz usando 3 bits, imagens com 16 cores usam 4 bits, imagens com 256 cores usam 8 bits, as imagens *true color* usam 3 bytes (RGB), e as imagens *true color* com transparência usam 4 bytes (Alpha + RGB), onde Alpha representa uma mascara para a transparência da imagem.

Existem dezenas de formatos de armazenamento de imagens em disco. Estes formatos foram criados para padronizar a estrutura da imagem gravada em disco e possibilitar o seu intercâmbio entre diferentes aplicações.

A grande variedade dos formatos de armazenamento de imagens se justificam pela necessidade de eficiência no armazenamento da imagem em disco, no tempo de leitura da imagem, na possibilidade de armazenamento da imagens com múltiplos níveis de detalhamento, e na inclusão de metadados.

Imagens de grandes dimensões, como as imagens de satélite de alta resolução, geram matrizes de pontos que consomem grande quantidade de memória para serem representadas. Estas imagens podem ser compactadas para economizar espaço de armazenamento, e incluir metadados para o georreferenciamento da imagem na superfície terrestre.

As imagens usadas na área médica, utilizam uma grande quantidade de cores para obter melhor detalhe nas imagens geradas, e também incluem metadados com informações sobre o paciente e o exame realizado.

Portanto, cada formato de imagem possui características específicas do seu campo de aplicação, e em memória são sempre representados por matrizes de pontos [19].

4.3.1 SIMPLIFICAÇÃO DE IMAGENS *RASTERS*

O processo de simplificação de imagens *rasters*, é necessário para otimizar a transferência das imagens entre servidores e estações clientes. Este processo, procura reduzir a quantidade de pontos por polegada da imagem, reduzir a área da imagem, ou reduzir a quantidade de cores usada na representação de cada ponto.

Este processo é utilizado por servidores de imagens para permitir que imagens com alto nível de detalhamento sejam simplificadas, com o objetivo de reduzir o seu tamanho e o nível de detalhe antes da transferência para a estação cliente.

A Figura 4.9, apresenta uma ortofoto do município do Rio de Janeiro, em escala inicial de 1/25.000, com 14.300 x 15.100 *pixels*, 864 MB de espaço ocupado em disco, e com precisão do *pixel* de 35 cm [19].

Figura 4.9: Ortofoto em escala 1/25.000, com 14.300 x 15.100 *pixels*, e 864 MB.

A Figura 4.10, apresenta o resultado da simplificação da mesma ortofoto, para uma escala final de 1/200.000, com 1.787 x 1.887 *pixels*, 14 MB de espaço ocupado em disco, e com precisão do *pixel* de 280 cm.

Figura 4.10: Ortofoto em escala 1/200.000, com 1.787 x 1.887 *pixels*, e 14 MB.

Imagens que representam áreas muito extensas podem ser recortadas para serem visualizadas na tela com um nível de detalhamento menor e processando mais rápido porque ocorrerá somente na região recortada, reduzindo significativamente a quantidade de dados transferidos.

4.3.2 CONVERSÃO DE FORMATO DE IMAGENS *RASTERS*

Devido a variedade de formatos, precisamos efetuar conversões para obter interoperabilidade entre diferentes dispositivos.

Por exemplo, os equipamentos de raio-x e ultrassonografia atuais geram imagens no formato DICOM que possuem metadados e são manipulados por sistemas de gerenciamento de imagens de pacientes (PACS). Para visualização das imagens DICOM em um navegador de Internet é necessário converter a imagem em uma imagem JPEG.

A Tabela 4.2, apresenta uma lista com os principais formatos de imagens *rasters* suportados pela biblioteca GDAL, usada pelos módulos de processamento de imagem [19].

Long name
Arc/Info ASCII Grid
ACE2
ADRG/ARC Digitized Raster Graphics (.gen/.thf)
Arc/Info Binary Grid
AIRSAR Polarimetric Format
Azavea Raster Grid
Bathymetry Attributed Grid
Magellan BLX Topo File Format
Microsoft Windows Device Independent Bitmap
Better Portable Graphics
Maptech/NOAA BSB Nautical Chart Format
VTP .bt Binary Terrain Format
Natural Resources Canada's Geoid file format (.byn)
AutoCAD DWG raster layer
CALS Type 1
CEOS Image
DRDC COASP SAR Processor Raster
Cloud Optimized GeoTIFF generator
TerraSAR-X Complex SAR Data Product
Convair PolGASP data
CTable2 Datum Grid Shift
USGS LULC Composite Theme Grid
DAAS (Airbus DS Intelligence Data As A Service driver)
DB2 raster
DirectDraw Surface
Derived subdatasets driver

Long name
Spot DIMAP
ELAS DIPEx
OPeNDAP Grid Client
First Generation USGS DOQ
New Labelled USGS DOQ
Military Elevation Data
Arc/Info Export E00 GRID
ECRG Table Of Contents (TOC.xml)
Enhanced Compressed Wavelets (.ecw)
Google Earth Engine Data API Image
ESRI .hdr Labelled
Erdas Imagine Raw
Earth Resources Laboratory Applications Software
ENVI .hdr Labelled Raster
Wavelet compressed images
ERMapper .ERS
Envisat Image Product
Esri Compact Cache
Extended Dynamic Range Image File Format
EOSAT FAST Format
FIT
Flexible Image Transport System
Fuji BAS Scanner Image
Generic Binary (.hdr labelled)
Oracle Spatial GeoRaster
Sandia National Laboratories GSAT File Format
Graphics Interchange Format
GMT Compatible netCDF
GeoPackage raster
GRASS Raster Format
GRASS ASCII Grid
WMO General Regularly-distributed Information in Binary form
Golden Software Surfer 7 Binary Grid File Format
Golden Software ASCII Grid File Format
Golden Software Binary Grid File Format
GSC Geogrid
Generic Tagged Arrays
GeoTIFF File Format

Long name
Grid eXchange File
Hierarchical Data Format Release 4 (HDF4)
Hierarchical Data Format Release 5 (HDF5)
ISO/IEC 23008-12:2017 High Efficiency Image File Format
HF2/HFZ heightfield raster
Erdas Imagine .img
Image Display and Analysis
Idrisi Raster Format
IGN France height correction ASCII grids
Raster Map
Intergraph Raster Format
Vaisala's weather radar software format
ISCE
International Service for the Geoid
USGS Astrogeology ISIS Cube (Version 2)
USGS Astrogeology ISIS Cube (Version 3)
Japanese DEM (.mem)
ERDAS JPEG2000 (.jp2)
JPEG-2000 (based on Kakadu)
JPEG2000 driver based on Lurawave library
JPEG2000 via MrSID SDK
JPEG2000 driver based on OpenJPEG library
JPEG JFIF File Format
Implementation of the JPEG-2000 part 1
JPEGLS
JPIP Streaming
KEA
KML Superoverlay
KOLOR Raw format
NOAA Polar Orbiter Level 1b Data Set (AVHRR)
Erdas 7.x .LAN and .GIS
FARSITE v.4 LCP Format
Daylon Leveller Heightfield
NADCON .los/.las Datum Grid Shift
OziExplorer .MAP
Meta Raster Format
MBTiles

Long name
In Memory Raster
Vexcel MFF Raster
Vexcel MFF2 Image
MrSID/MG4 LiDAR Compression / Point Cloud View files
Multi-resolution Seamless Image Database
Meteosat Second Generation
Meteosat Second Generation (MSG) Native Archive Format (.nat)
NLAPS Data Format
NetCDF: Network Common Data Form
NOAA NGS Geoid Height Grids
NextGIS Web
National Imagery Transmission Format
NTv1 Datum Grid Shift
NTv2 Datum Grid Shift
Northwood/Vertical Mapper File Format
Northwood/Vertical Mapper File Format
OGC API Tiles / Maps
OZF2/OZFX3 raster
JAXA PALSAR Processed Products
PCI .aux Labelled Raw Format
PCI Geomatics Database File
PCRaster raster file format
Geospatial PDF
Planetary Data System v3
NASA Planetary Data System (Version 4)
PLMosaic (Planet Labs Mosaics API)
Portable Network Graphics
Netpbm (.pgm, .ppm)
PostGIS Raster driver
PHOTOMOD Raster File
R Object Data Store
Rasdaman GDAL driver
Rasters in SQLite DB
Rasters in SQLite DB
RDA (DigitalGlobe Raster Data Access)
RIEGL Database

Long name
Swedish Grid Maps
Raster Matrix Format
ROI_PAC
Raster Product Format/RPF (a.toc)
R Raster
RadarSat 2 XML Product
Sentinel-1 SAFE XML Product
CEOS SAR Image
SAGA GIS Binary Grid File Format
USGS SDTS DEM
Sentinel-2 Products
SGI Image Format
Scaled Integer Gridded DEM
Snow Data Assimilation System
Standard Product Format (ASRP/USRP) (.gen)
SRTM HGT Format
Spatio-Temporal Asset Catalog Tiled Assets
Terragen™ Terrain File
TARGA Image File Format
EarthWatch/DigitalGlobe .TIL
TileDB
TerraSAR-X Product
USGS ASCII DEM (and CDED)
VICAR
GDAL Virtual Format
OGC Web Coverage Service
WEBP
Web Map Services
OGC Web Map Tile Service
X11 Pixmap
ASCII Gridded XYZ
ZMap Plus Grid

Tabela 4.2: Formatos de imagens *rasters* suportados pela biblioteca GDAL.

4.3.3 REPROJEÇÃO DE IMAGENS *RASTERS*

Os sistemas de geoprocessamento precisam apresentar os dados no sistema de coordenada desejado pelo usuário permitindo a utilização adequada da informação.

Desta forma, um sistema de localização veicular poderá receber os dados em coordenadas geográficas que facilitam a integração com equipamentos GPS. Enquanto, empresas de construção de pontes e barragens podem receber os mapas em coordenadas UTM que fornecem uma medição mais precisa.

A Figura 4.11a, apresenta uma ortofoto do município do Rio de Janeiro, em escala inicial de 1/25.000, de um aerolevantamento do IBGE em coordenadas geográficas, e a Figura 4.11b, apresenta o resultado da reprojeção da ortofoto no sistema de coordenadas e projeção UTM.

Figura 4.11: (a) Ortofoto em coordenadas geográficas, e (b) **r**eprojeção em coordenadas UTM.

Podemos observar que a reprojeção da imagem do sistema de coordenadas geográficas para o sistema de coordenadas e projeção UTM, provoca um distorção normal do processo de conversão, que pode ser observado pela faixa lateral escura.

O processo de reprojeção de imagens de coordenadas geográficas para UTM, também sofre distorções motivadas pela conversão do *pixel* que representa coordenadas reais usando um elemento discreto [19].

Capítulo 5

Recursos de Analise de Imagens *Rasters*

Este trabalho realiza um estudo sobre as unidades de armazenamento do tipo *computational storage devices* (CSD), de grande capacidade de armazenamento, e que possuem recursos de processamento embutido, *in-situ processing*.

A identificação de padrões nas imagens diretamente nos equipamentos de monitoramento e supervisão, reduz de forma significativa o volume de dados trafegados pela rede.

Neste estudo, foi implementado um mecanismo de armazenamento otimizado dos modelos de aprendizado de redes neurais aplicado na classificação de texturas, na detecção de mudanças, e no reconhecimento de padrões, explorando a capacidade de processamento das unidades de armazenamento com processamento embutido (CSD).

5.1 REDES NEURAIS SEM PESO WISARD

Este trabalho avalia o uso das redes neurais sem peso WiSARD no processo de classificação de texturas, detecção de mudanças, e no reconhecimento de padrões, comparando o tempo de treinamento e classificação das imagens, e a precisão obtida por este modelo de aprendizado de maquina com o modelo de redes neurais convolucionais (CNN) [42 e 43].

A principal características da rede neural WiSARD é a rapidez de processamento da fase de aprendizado e classificação do modelo. Porém, estes modelo de redes neurais consomem muita memória quando aplicado no processamento de imagens *rasters*.

As redes neurais sem peso realizam o mapeamento dos *bits* da imagem em um vetor de endereçamento de memória, que armazena o aprendizado obtido na fase de treinamento da rede neural.

A rede neural sem pesos WiSARD procura identificar a frequência de ocorrência dos padrões de *pixels* da imagem. Desta forma, a imagem é particionada em conjuntos de *pixels* de dimensões fixas, denominados de n-upla. Essa partição é formada aleatoriamente, e se mantém fixa por todo o ciclo de vida da rede neural, Figura 5.1 [44].

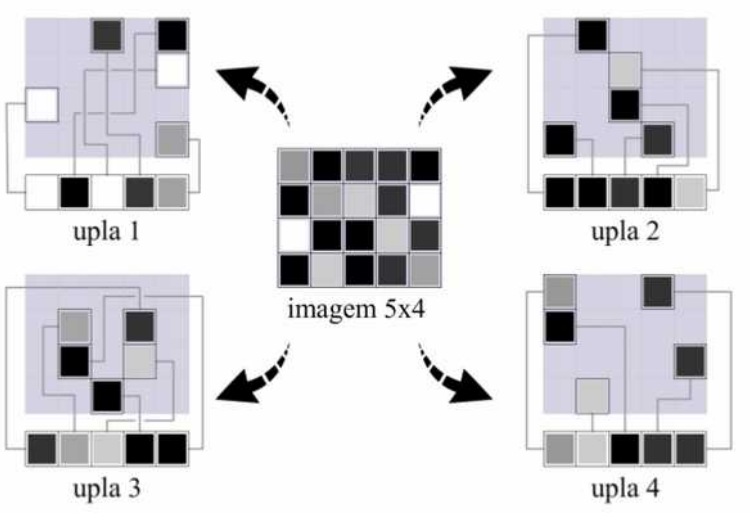

Figura 5.1: Particionamento da imagem em n-uplas de *pixels*.

Cada conjunto de n-upla de *pixels* é associada a um nó de memoria, onde cada endereço aponta para um 1 *bit* que armazena a ocorrência do padrão. Todas as n-uplas da imagem possuem o seu respectivo nó de memória, Figura 5.2 [44].

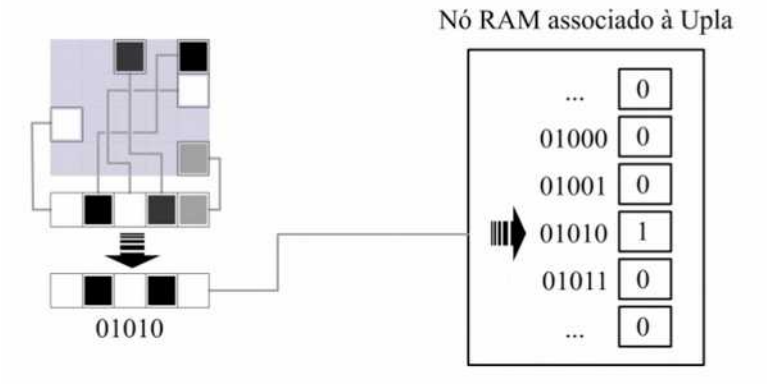

Figura 5.2: Nó RAM associado a n-upla.

No treinamento da rede neural sem pesos WiSARD, é definido um limiar a cada n-upla de *pixels*, convertendo essa n-upla em um vetor binário de n-upla *bits*. Esse vetor por sua vez é usado como um endereço para o nó RAM associado a essa n-upla.

O valor de um dígito binário para o qual esse endereço aponta é ajustado para 1. Esse processo é executado em cada nó RAM pertencente ao neurônio que corresponde à classe a ser treinada.

Na classificação da WiSARD, os endereços para os nós RAM são extraídos das diversas n-uplas de *pixels* que compõem a partição da imagem, da mesma maneira como é feito no caso do treinamento, e um contador em cada neurônio é zerado no início da operação de classificação da imagem. Para cada n-upla onde o endereço gerado aponta para um valor 1, este contador é incrementado.

O valor final desse contador constitui a resposta do neurônio a imagem apresentada, e vence o neurônio que apresentar a resposta mais alta, ou seja, o neurônio cujo contador tiver a soma mais alta.

A Figura 5.3, apresenta a arquitetura de classificação da rede neural sem pesos WiSARD, onde o contador de valor mais alto determina o neurônio com a resposta mais provável [44].

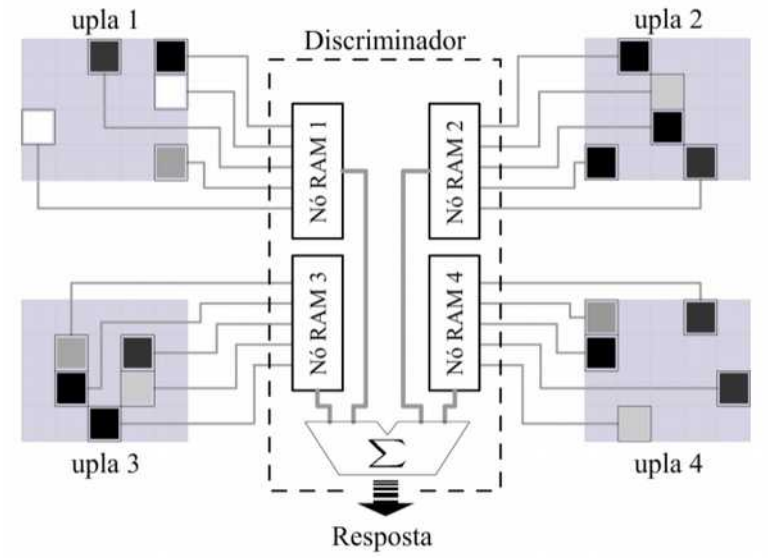

Figura 5.3: A arquitetura do neurônio WiSARD.

ImagePattern: Implementação da Rede Neural sem Pesos WiSARD

A implementação da rede neural sem peso WiSARD foi realizada usando um mapa de memória. As imagens de entrada são preprocessadas para reduzir o número de cores da imagem RGB (256 x 256 x 256 = 16.777.216 cores) para 256 cores, e agrupadas em faixas com 8 e 32 cores.

A Figura 5.4, apresenta o resultado após a redução da profundidade de cores da imagem RGB para 256 cores (1 byte), usado como entrada para a rede neural sem pesos WiSARD.

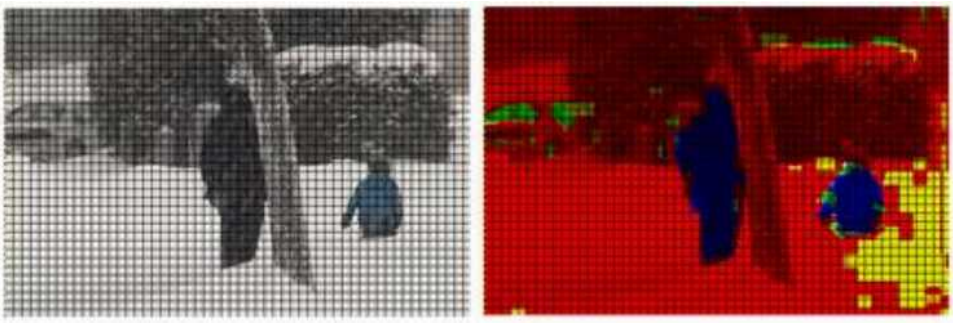

Figura 5.4: Imagem preprocessada em agrupamentos de 8 e 32 cores.

Este experimento comparou o desempenho do servidor de imagens, HORUS Image Server, na criação do modelo de aprendizado e na classificação das imagens usando com a rede neural sem pesos WiSARD, usando imagens separadas em duas categorias de processamento:

i) 8 faixas de cores por pixel (32 cores por faixa = 1 byte)
ii) 32 faixas de cores por pixel (8 cores por faixa = 1 byte)

Em seguida, cada pixel da imagem é usado como uma n-upla de dimensões 8 e 32 *bits*, de acordo com o número de faixas, isto é, com respectivamente 32 ou 8 cores por faixa, e a n-upla gerada por cada pixel é usada para mapeamento do nó RAM da WiSARD, conforme podemos observar na Figura 5.5.

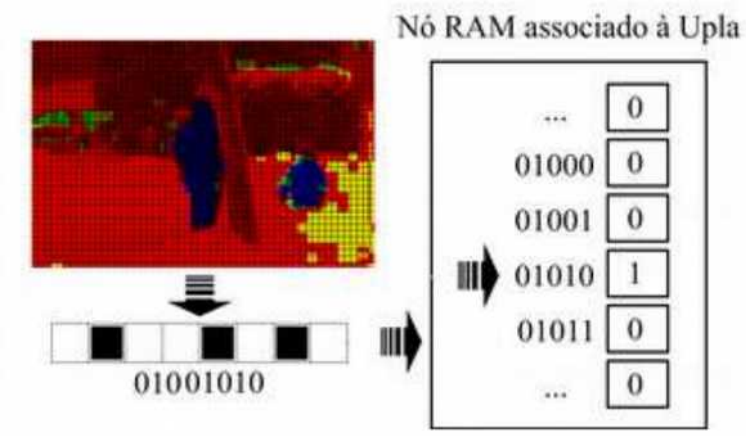

Figura 5.5: Agrupamento RGB em n-upla de 8 *bits*, associada a um nó RAM.

O nó RAM possui um contador associado para cada uma das n-uplas mapeadas nos *pixels* das imagens. Este contador é incrementado a medida que um mesmo pixel da imagem apresenta o mesmo padrão.

A operação de impressão da sequencia de imagens de vídeo nos nós RAM da WiSARD, é similar a captura de imagens por câmeras fotográficas com longo período de exposição. Esta operação corresponda a fase de treinamento da rede neural sem pesos WiSARD, e para este experimento, usamos 90% das imagens do *dataset* nesta atividade.

Após a fase de treinamento da rede neural sem pesos WiSARD, aplicamos o modelo na classificação das imagens separadas para uso pela fase de teste, isto é, 10% da imagens do *dataset*.

Nesta fase, os *pixels* da imagem de teste são convertidos de RGB para 8 e 32 faixas de cores, e usado para endereçamento do nó RAM, e em seguida, em função do valor do contador, decidimos se o pixel corresponde ao *background* original da imagem.

Este procedimento produz uma imagem em monocromática (preto e branco), onde cada *pixel* ativo corresponde a um pixel com padrão diferença ao *background* que foi identificado na fase de treinamento. A Figura 5.6, apresenta o resultado final obtido após o processamento da imagem usando a rede neural sem pesos WiSARD.

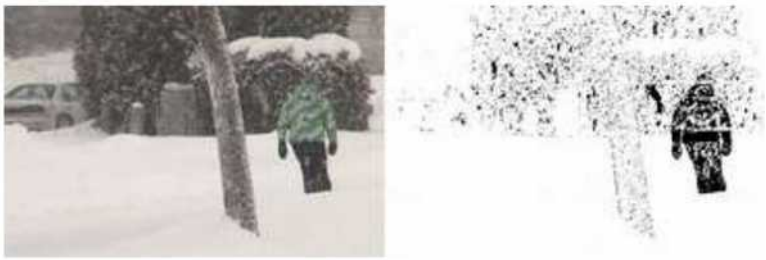

Figura 5.6: Resultado final obtido após processamento usando WiSARD.

Como as imagens do vídeo possuem variações de tonalidades, para cada imagem é calculado o percentual de *pixels* que sofreram variações em relação ao *background* original esperado. Este valor é comparado com um limiar (*threshold*), e se for maior, consideramos que na imagem existem objetos relevantes em movimento.

A rede neural sem peso WiSARD é usada somente na fase de detecção de mudanças e identificação do *background* da imagem. A identificação dos objetos em movimento, é resolvida usando a análise de *clusters* K-Means na imagem em preto e branco.

Analise de *Clusters* K-Means

O algorítimo K-Means permite o agrupamento de N pontos em K conjuntos, denominados *clusters*, onde os elementos contidos em cada *cluster* são agrupados em função da similaridade existente entre eles.

No caso do agrupamento de pontos em um plano cartesiano, o *cluster* será definido por um elemento central, denominado centroide, e os pontos pertencentes ao *cluster*, serão os mais próximos do centroide.

O problema de *clustering* possui complexidade $O(N^{dk+1})$, e para realização da análise de *clusters*, a imagem gerada pela rede neural sem pesos WiSARD é dividida em regiões, Figura 5.7. Neste experimento, definimos regiões com dimensões de 54 x 36 *pixels*, onde para cada região é estabelecido um valor referente ao percentual de *pixels* ativos existentes.

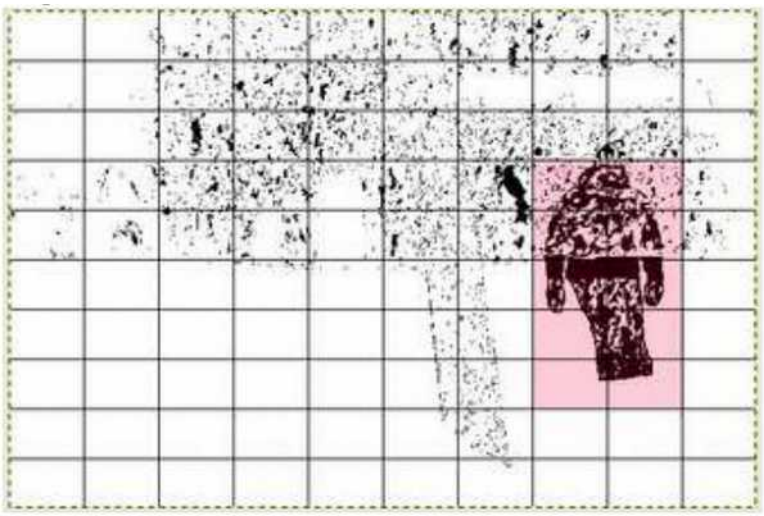

Figura 5.7: Identificação da região de movimento.

Em seguida, a análise de *clusters* identifica as regiões com maior aglomeração de pontos, e atribui até 10 centroides à imagem, e as regiões mais próximas dos centroides são destacadas na imagem final, conforme observamos na Figura 5.7.

Algorítimo de Analise de *Clusters* AlgorKMean

Neste estudo, implementamos o algorítimo AlgorKMean para realização da análise de *clusters* usando um processo similar ao algorítimo K-Means. Este algorítimo não possui capacidade de processamento de imagens de grandes dimensões, que estão armazenadas de forma distribuída em um *cluster* de computadores.

Porém, é bastante adequado para o processamento de imagens armazenados inteiramente em um único nó de processamento, e a implementação deste algorítimo possibilitou comparar o desempenho do servidor de imagens, HORUS Image Server, na identificação de mudanças em imagens.

Para realização da análise de *clusters*, a imagem de entrada é dividida em regiões com dimensões de 54 x 36 *pixels*, sendo calculado o valor percentual de *pixels* ativos em cada região.

As regiões com maior concentração de *pixels* ativos são identificadas com *clusters* da imagem, e as regiões vizinhas são analisadas e de acordo com a distância e concentração de pontos existentes, e são selecionadas na imagem, identificando a área com maior possibilidade de haver um objeto em movimento, Figura 5.8.

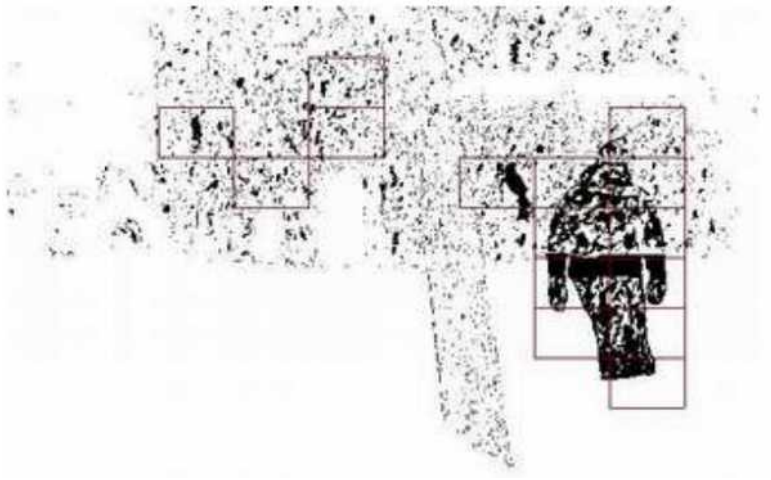

Figura 5.8: Identificação das regiões com possibilidade de movimento.

O processo final consiste em escolher as áreas que possuem maior possibilidade de haver movimento, em função de um limiar (*threshold*) definido.

5.2 REDES NEURAIS PROFUNDAS (DNN)

As redes neurais profundas, *deep neural networks* (DNN), são modelos de redes neurais fortemente conectadas que apresentam uma camada de entrada, duas ou mais camadas ocultas e uma camada de saída. A Figura 5.9, apresenta um modelo de redes neurais profundas (DNN) [26, 27 e 28].

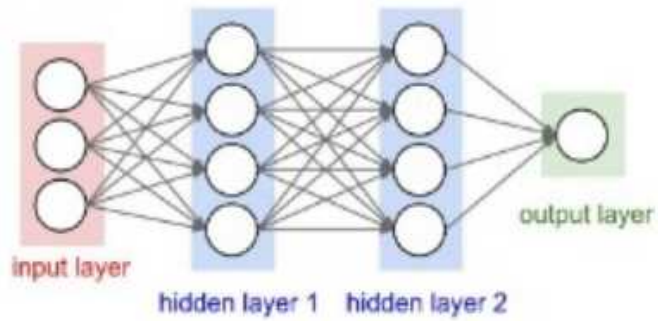

Figura 5.9: Modelo de redes neurais profundas.

As redes neurais convolucionais, são modelos de redes neurais profundas que são largamente utilizadas na classificação de imagens e áudio. Este modelo de rede neural procura separar em cada camada as características que definem as imagens, permitindo a identificação dos dados. A Figura 5.10, apresenta um modelo de rede neural convolucional aplicada ao reconhecimento de imagens [30, 31 e 32].

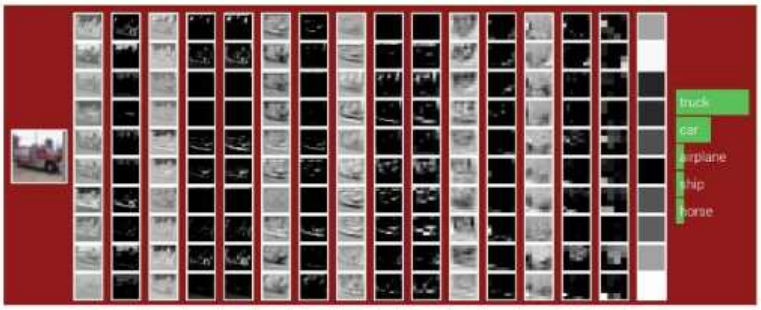

Figura 5.10: Modelo de redes neurais convolucionais.

Redes Neurais Convolucionais

Ao analisarmos imagens de 256 cores usando redes neurais, podemos arrumar os elementos da imagem em um vetor. Desta forma, uma imagem de 28 x 28 pixels e 256 cores, seria representada por um vetor com 784 bytes, e utilizaríamos na camada inicial da rede neural 784 entradas, Figura 5.11.

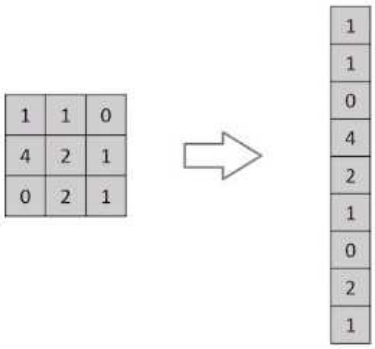

Figura 5.11: Organização dos *pixels* da imagem em vetor.

Quando estudamos imagens com maior quantidade de cores, como imagens RGB, o tamanho do vetor necessário para representar a imagem é de 2.352 bytes. Além disso, ao representar uma imagem por um vetor de dados, perdemos as características associadas a cada profundidade de cor, gerando um modelo com menor taxa de assertividade [33].

Desta forma, sugiram as redes neurais convolucionais, que separam os dados de acordo com a profundidade de cor aplicada as imagens, assegurando um melhor desempenho na classificação das imagens, Figura 5.12.

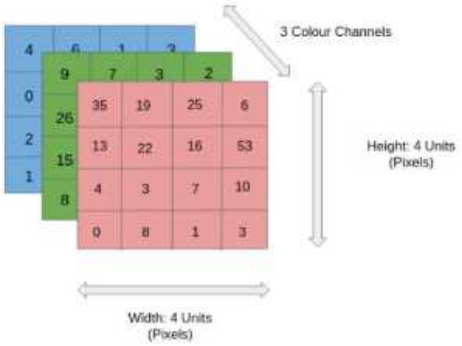

Figura 5.12: Organização dos *pixels* da imagem por canal de cor.

Camadas Convolucionais (*Kernel*)

Cada camada convolucional do modelo de redes neurais implementa uma função de filtro denominada *kernel*, que procura extrair os detalhes da imagem.

Esta operação é realizada na imagem e reduz a quantidade de informações para um número específico de elementos. A Figura 5.13 apresenta o processo de filtro aplicado por uma camada convolucional [33 e 34].

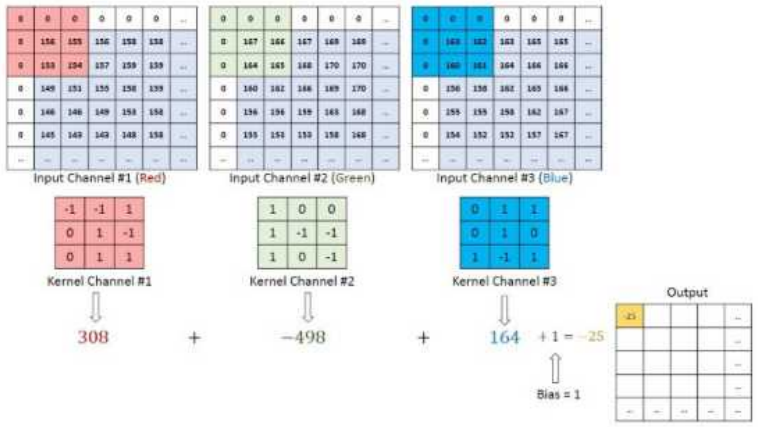

Figura 5.13: Processo de filtro da aplicação pela camada de convolução.

O filtro é aplicado em cada canal de cor, e o resultado obtido da camada de convolução é uma matriz com um número reduzido de elementos. No processo de convolução, as operações mais comuns aplicadas pelos filtros, são operações de identificação de valores médios ou máximos da camada de entrada.

5.3 REDES NEURAIS PROFUNDAS DISTRIBUÍDAS (DDNN)

O aumento na capacidade de processamento das unidades de monitoramento permite que sejam realizadas classificações nos dados coletados nos dispositivo remotos, possibilitando a decisão sobre o envio ou não da imagem.

A pré-classificação das imagens realizada pelos próprios equipamentos de monitoramento e supervisão, reduz em até 20x o volume dos dados trafegados pela rede. Deste modo, muito esforço tem sido aplicado para implementar em microcontroladores com pouca capacidade de armazenamento capacidade de classificação usando redes neurais, e deste esforço surgiram os conceitos sobre (i) TinyML e (ii) *Federated Learning* (FL) [36, 37 e 38].

i) TinyML

O conceito de TinyML tem como base a otimização do modelo de redes neurais, através da redução da quantidade de parâmetros do modelo e da transformação dos valores dos parâmetros de *floating-point* para valores inteiros ou em alguns casos em valores de 0 a 256 (1 byte).

Isto se torna necessário, porque o modelo de TinyML é usado em dispositivos com processadores de baixo custo e com baixa capacidade de processamento, pouca memória RAM para execução dos programas, e espaço de armazenamento de dados limitado. A Tabela 5.1, apresenta a especificação dos microcontroladores ATMega328 de 8 bits, 20 MHz, 2 KB SRAM, e 32 KB de *Flash Memory,* e o PIC16F873A de 8 bits, 20 MHz, 192 bytes RAM, e 4 KB *Flash Memory* [38].

ATMega328 . Low Power 8-Bit Microcontroller . Advanced RISC Architecture – 32 x 8 General Purpose Working Registers – Up to 20 MIPS Throughput at 20 MHz – On-chip 2-cycle Multiplier . High Endurance Non-volatile Memory Segments – 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory – 256/512/512/1K Bytes EEPROM – 512/1K/1K/2K Bytes Internal SRAM . Price: US$ 1.11/unit to US$ 1.21/unit . Reference: Alibaba.com in 4th September, 2022.	Microchip PIC16F873A . Low Power 8-Bit Microcontroller . Instruction Set - 35 Instructions . Operating Frequency - DC – 20 MHz . Flash Program Memory (14-bit words) – 4 KB . Data Memory - 192 bytes . EEPROM Data Memory - 128 bytes . Price: US$ 2.00/unit to US$ 5.00/unit . Reference: Alibaba.com in 4th September, 2022.

Tabela 5.1: Microcontroladores de baixo custo ATMega328 e PIC16F873A.

ii) *Federated Learning* (FL)

O aprendizado federado, *federated learning* (FL), é um processo de aprendizado de máquina que explora a capacidade de comunicação de dados dos dispositivos remotos e distribui o processamento de dados para construção do modelo de aprendizado.

No aprendizado federado, uma rede de sensores coleta os dados e encaminha para um microcontrolador que concentra os dados recebidos dos sensores. Nestes dispositivos remotos, um processo atualiza periodicamente um modelo parcial de aprendizado de máquina, com base nos dados coletados por cada microcontrolador.

Após a atualização do modelo parcial, os parâmetros são compartilhados entre todos os dispositivos e são utilizados para aprimorar a precisão do modelo global de aprendizado de máquina.

Deste modo, o processo de criação do modelo global é distribuído entre os dispositivos remotos, reduzindo a carga de trabalho realizada por cada dispositivo. A Figura 5.14, apresenta a arquitetura do modelo de aprendizado federado [37].

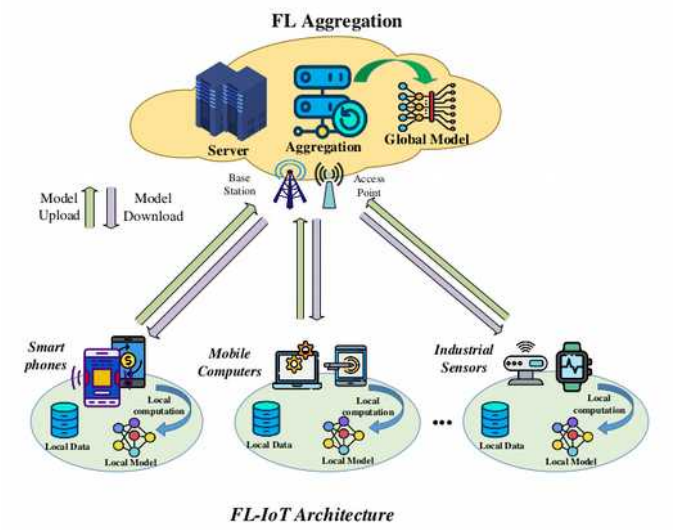

Figura 5.14: Arquitetura do modelo de aprendizado federado.

As bibliotecas TinyML possibilitaram a aplicação do aprendizado federado em qualquer dispositivo remoto, como computadores, *tablets*, *smartphones* e microcontroladores de baixo custo.

5.4 DETECÇÃO DE *BACKGROUND*

O processo de detecção de *background* é fundamental para isolar os elementos em movimento em uma imagem. Após a identificação do *background*, conseguimos identificar melhor as mudanças em uma imagem.

Para avaliação do servidor de imagens, HORUS Image Server, usamos o *dataset Change Detaction DATASET* 2014 (CDnet2014), que pode ser obtido no endereço: http://changedetection.net/ [44, 45 e 46].

Este *dataset* é específico para avaliação de algorítimos de detecção de *background*, e contém 11 categorias de vídeos com 4 a 6 sequencias de vídeo em cada categoria, e cada arquivo de vídeo é constituído por 1 diretório com uma sequencia de imagens no formato JPEG.

Esta imagem define através de uma mascará de pontos os elementos do *background*, e contém 5 áreas identificadas: *static*, *hard shadow*, *outside region of interest*, *unknown motion*, *motion*. A Figura 5.15, apresenta uma sequencia de imagens obtida do *dataset Skating*.

Figura 5.15: Sequencia de imagens do *dataset Skating*.

Neste experimento usamos um subconjunto de imagens do *Change Detection Dataset* 2014. (i) *Bus Station* com 1.250 imagens, (ii) *Pedestrians* com 1.100 imagens, e (iii) *Skating* com 3.900 imagens.

A Figura 5.16, apresenta algumas imagens obtidas de cada uma das categorias selecionadas do *dataset* usado neste experimento: *Bus Station*, *Pedestrians*, e *Skating*.

Figura 5.16: Imagens do *Benchmark*: *Bus Station*, *Pedestrians*, Skating.

Definição dos Dados para o *Benchmark*

O *Benchmark* avalia três conjuntos de dados do *dataset* CDnet2014. *Bus Station* (1.250 imagens), *Pedestrians* (1.100 imagens) e *Skating* (3.900 imagens), e compara duas configurações diferentes da aplicação. A primeira, usa 8 faixas de cores na detecção de mudanças, e a segunda usa 32 faixas de cores.

Implementação do Módulo de Classificação – *Change Detection*

Neste módulo, as imagens são recortadas em regiões de 5.000 m x 5.000 m, e os recortes consideram uma origem comum para as coordenadas da imagem, Figura 5.17.

As imagens utilizadas estão no formado UTM-23S / SIRGAS 2000, e cada recorte da imagem será comparado com versões de anos anteriores da mesma região, usando um modelo de aprendizado que irá classificar o que representa *background* dos novos objetos que surgiram.

As regiões que limitam os novos objetos identificados serão armazenadas no banco de dados e poderão ser consultadas.

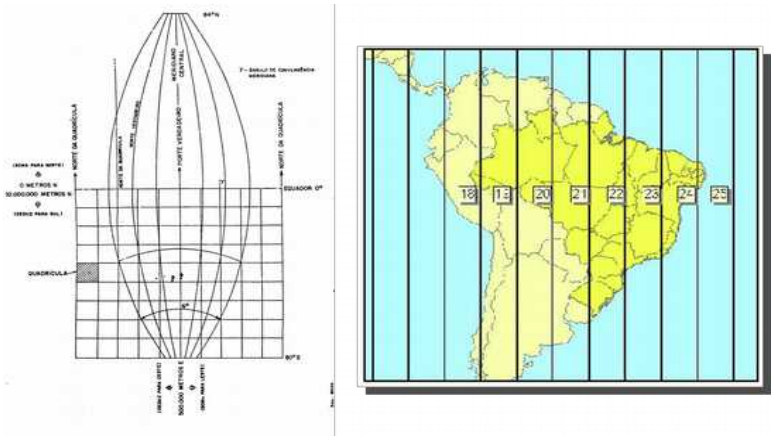

Figura 5.17: Recorte das regiões em faixas de 5.000 m x 5.000 m.

5.5 DETECÇÃO DE MUDANÇAS

O processo de detecção de mudanças em imagens possui diversas aplicações praticas, como por exemplo, o monitoramento em tempo real de áreas de segurança e a detecção automática das mudança ocorridas na vegetação nativa de uma região ao longo do tempo, Figura 5.18. Este processo vem sendo estudado de forma intensiva usando Redes Neurais, obtendo um ganho significativo de desempenho na classificação dos dados [46].

Figura 5.18: Exemplo de identificação de mudanças.

5.6 CLASSIFICAÇÃO DE TEXTURAS EM IMAGENS

O problema de classificação de texturas em imagens é fundamental para o reconhecimento de padrões locais que se repetem em uma área maior da imagem. A identificação e classificação destes padrões locais, permite o reconhecimento de áreas da imagem que são cobertas por um mesmo padrão de textura.

Nosso objetivo e implementar um mecanismo de classificação de texturas otimizado usando Redes Neurais que permita classificar rapidamente as áreas cobertas por cada tipo de textura presente nas imagens.

Este tipo de processamento possui diversas aplicações praticas, como por exemplo, a classificação automática de diferentes tipos de solos e diferentes tipos de vegetação existentes em uma imagem de satélite, Figura 5.19 [46].

Figura 5.19: Exemplo de classificação de texturas.

5.7 IDENTIFICAÇÃO DE ELEMENTOS EM IMAGENS

O reconhecimento de padrões em imagens é um tema que vem sendo amplamente estudado e aplicado no reconhecimento de escrita, no reconhecimento facial, e etc. A figura abaixo, apresenta um exemplo de aplicação de reconhecimento de padrões em imagens [30, 31, e 32].

Neste estudo, analisaremos o uso de Redes Neurais no reconhecimento de padrões em imagens, com o objetivo de implementar de forma eficiente este modelo de Redes Neurais com a expectativa de obter uma melhora significativa no treinamento do modelo e na classificação dos dados, Figura 5.20.

Figura 5.20: Exemplo de identificação de elementos em imagens.

Capítulo 6

Análise dos Resultados

Durante as fases de desenvolvimento e de experimentos com o servidor de imagens, HORUS Image Server, não tive acesso as unidades de armazenamento com capacidade de processamento embutido, *computational storage devices* (CSD), da empresa NGD Systems. Desta forma, o servidor de imagens foi desenvolvido, testado, e os experimentos foram conduzidos em um ambiente simulado.

Realizamos três experimentos usando um ambiente simulado com 1 Notebook SAMSUNG, Modelo NP550PSC-AD1BR, com Processador Intel Core i7-3630QM de 2,40 GHz, com 4 Núcleos de Processamento e suporte para até 8 *Threads*, 8 GB de Memória RAM, e disco rígido convencional de HDD 1.0 TB.

Nos experimentos usamos o sistema operacional Linux CentOS 7, com o Kernel 3.10.0-1160 de 64 bits, o compilador GCC 10.2.0, e as bibliotecas SQLite 3.32.3, PROJ 6.3.2, GDAL 3.1.1.

As consultas foram preparadas para reproduzir um volume de dados similar as consulta apresentadas no artigo "*POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database*" [14].

Deste modo, o *Benchmark* foi preparado para ser executado em um ambiente simulado com com até 4 unidades de armazenamento CSD, Newport CSD da empresa NGD Systems, executando 6 diferentes consultas que recuperam os seguintes volumes de dados do servidor: TS-1 1,25%, TS-2 15,17%, TS-3 30.34%, TS-4 54,04%, TS-5 63.22%, e TS-6 100.00% [14].

Para reduzir o tempo de processamento dos experimentos no ambiente simulado, trabalhamos somente com a consulta TS-1, que recupera 4,7% dos dados e processa cerca de 190 imagens.

Escolhemos para o *Benchmark* as operações mais frequentes em sistemas de geoprocessamento, como a conversão de imagens GeoTIFF para JPEG, a projeção das imagens do sistema de coordenadas geográficas para o sistema de coordenadas UTM, e a simplificação das imagens para ¼ do tamanho original.

6.1 MÉTRICAS UTILIZADAS

As métricas básicas adotadas neste experimento foram tempo de processamento, aceleração (*speedup*), media aritméticas e media harmônica.

i) O tempo de processamento é medido em mile segundos (ms).

T_{proc} (Equação 6.1)

ii) A aceleração (*speedup*) é a razão entre o tempo de processamento da configuração de referência T_{base} e o tempo de processamento do experimento T_{proc}.

$$\text{Aceleração} = T_{base} / T_{proc} \quad \text{(Equação 6.2)}$$

iii) Os valores médios apresentados neste capítulo são fornecidos por médias aritméticas e harmônicas dos dados conforme as expressões à seguir.

(a) Média aritmética é obtida da expressão.

$$AM = \left(\sum_{i=0}^{n} S_i \right) / n \qquad \text{(Equação 6.3)}$$

Onde n - total de valores computados.
S_i - valores computados.

(b) Média harmônica é obtida da expressão.

$$HM = n \left(\sum_{i=0}^{n} 1/S_i \right)^{-1} \qquad \text{(Equação 6.4)}$$

Onde n - total de valores computados.
S_i - valores computados.

6.2 RESULTADOS OBTIDOS EM AMBIENTE SIMULADO

i) Primeiro Experimento (Experimento Inicial)

Os experimentos iniciais foram conduzidos usando a versao do sistema HORUS Image Server v1.0.20210714. Nesta versão a distribuição dos dados era realizada no nível dos arquivos, e a unidade remota que possuía o arquivo consultado era responsável por efetuar os processamentos de conversão, projeção e simplificação da imagem, e depois enviar a resposta ao servidor hospedeiro.

Nos experimentos iniciais simulamos até 4 unidades de armazenamento CSD, e obtivemos um ganho potencial em aceleração (*speedup*) de até 1,69 no processo de conversão de imagens GeoTIFF para JPEG (TS1-C), de até 1,20 na projeção das imagens que estão no sistema de coordenadas geográficas para o sistema de coordenadas UTM (TS1-R), e de até 1,48 na simplificação das imagens *rasters* (TS1-S).

As Figuras 6.1 e 6.2, apresentam os gráficos de tempo de processamento e de aceleração (*speedup*) obtidos no experimento inicial usando um ambiente simulado com 1 unidade de armazenamento SSD (SEQ), com 2 unidades de armazenamento CSD (U2), e com 4 unidades de armazenamento CSD (U4).

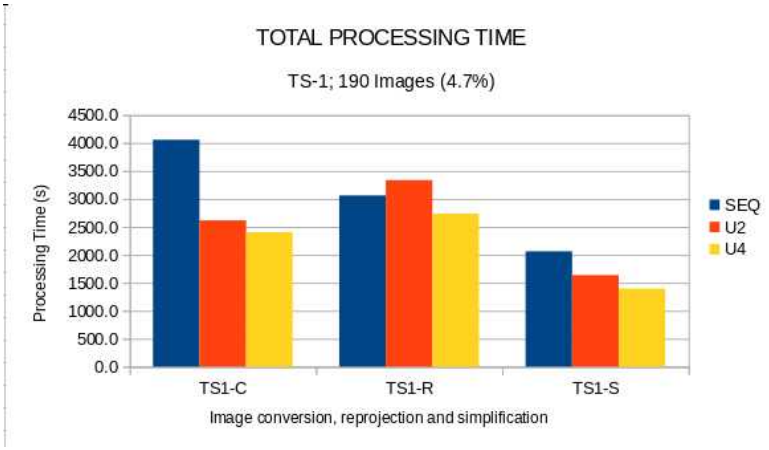

Figura 6.1: Comparação do tempo de processamento (Experimento Inicial).

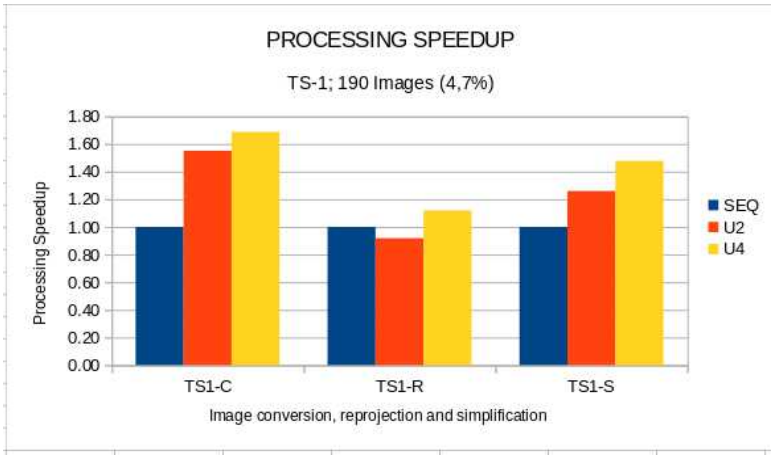

Figura 6.2: Comparação do *speedup* (Experimento Inicial).

Apos os primeiros experimentos em ambiente simulado, pude observar o ganho potencial que podemos obter com o uso das unidades de armazenamento com processamento embutido, *computational storage drives* (CSD).

Neste primeiro experimento, observei também o ganho potencial com a separação do processamento entre diferentes unidades de armazenamento CSD. Este ganho de desempenho pode ser obtido usando múltiplas unidades de armazenamento com capacidade de processamento embutido em sistemas de monitoramento, classificação e processamento por imagens.

ii) Segundo Experimento

Na segunda versão do sistema HORUS Image Server v2.1.20210804, corrigimos uma das deficiências presentes na primeira versão do sistema, que era a distribuição do armazenamento por arquivo.

Deste modo, aproveitamos a elevada taxa de transferência do barramento PCI-e e do protocolo NVMe, para distribuir os dados no nível de blocos entre as unidades de armazenamento CSD. A Figura 6.3, apresenta a separação dos dados no nível de blocos.

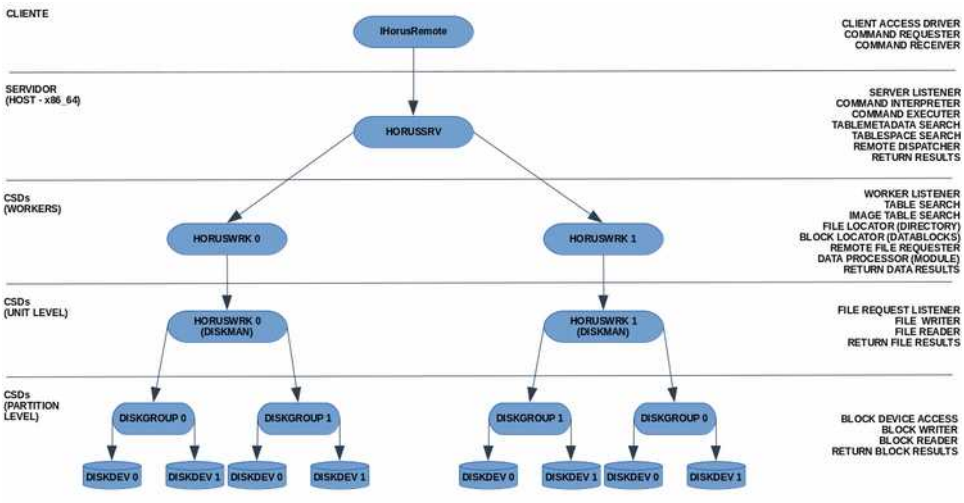

Figura 6.3: Distribuição dos dados no nível de blocos – HORUS v2.1.20210804.

iii) Terceiro Experimento

Porem, ainda havia uma outra deficiência no modelo proposto, que era o uso do modelo mestre escravo, e a nova versão do sistema HORUS Image Server v2.2.20220114 corrigiu, permitindo que diferentes unidades de armazenamento CSD recebam os dados e processem de forma distribuída as imagens.

A Figura 6.4, apresenta a separação dos dados no nível de blocos, e o processamento dos dados distribuído entre as unidades de armazenamento CSD.

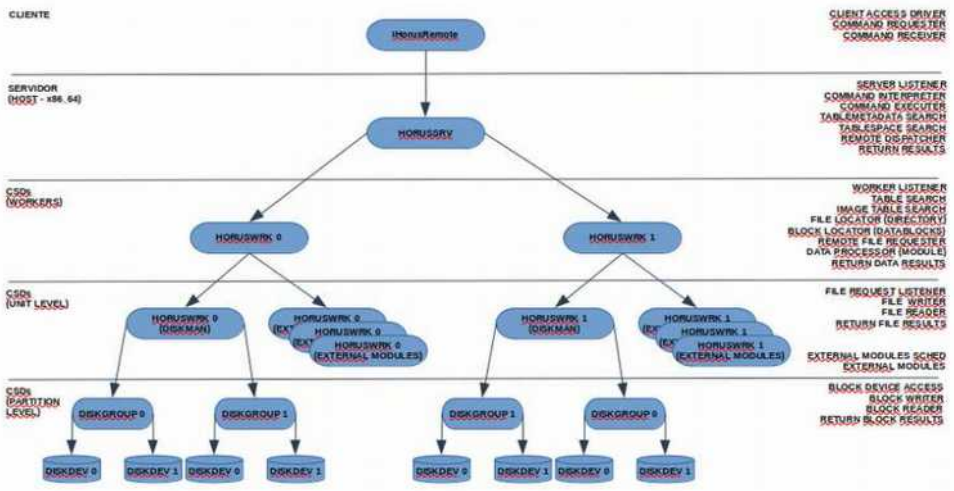

Figura 6.4: Distribuição dos dados no nível de blocos – HORUS v2.6.20220417.

A partir deste experimento pude observar que o tempo de processamento da criação da estrutura de diretórios e desprezível para o *Benchmark*, e que as operações mais sensíveis são as de *upload* dos dados e a de *download* e processamento.

6.3 DISTRIBUIÇÃO DO TEMPO DE EXECUÇÃO

O *Benchmark* usado contém 3 estágios de execução:

i) MKDIR: Estágio de criação dos diretórios.
ii) UPLOAD: Estágio de envio dos arquivos de imagens para o servidor.
iii) PROC: Estágio de processamento das imagens.

Conforme comentamos anteriormente, o tempo de criação da estrutura dos diretórios, estágio MKDIR, representa menos de 0,00001% do tempo de execução e será desprezado nas futuras análises.

A Tabela 6.1, apresenta os tempos de processamento de cada fase de execução do *Benchmark*. Nesta tabela separamos os resultados em 3 categorias: 1 unidade de armazenamento CSD (1CSD), 2 unidades de armazenamento CSD (2CSD), e 4 unidades de armazenamento CSD (4CSD).

Nesta tabela, separamos as configurações em armazenamento e processamento das imagens em disco, na região do *Datastore* (DS), ou em memoria compartilhada, usando *Shared Memory* (SM).

Além disso, separamos as configurações em *Single Threads* (ST) ou *Multi Threads* (MT) nas operações de leitura e escrita de dados no nível de blocos.

RESULTADOS

		MKDIR	UPLOAD	PROC
1CSD	DS-ST-1C	0.0009	3323.4447	11066.9752
	SM-ST-1C	0.0008	3453.3635	12179.4899
	DS-MT-1C	0.0009	3474.8353	11421.6605
	SM-MT-1C	0.0009	3123.2359	10797.4683
2CSDs	DS-ST-2C	0.0008	2616.5962	7655.1958
	SM-ST-2C	0.0011	2137.3474	6862.8875
	DS-MT-2C	0.0012	2202.4704	7402.6248
	SM-MT-2C	0.0005	1920.3308	6879.6718
4CSDs	DS-ST-4C	0.0041	1201.7843	5003.0896
	SM-ST-4C	0.0026	1305.6174	5180.0557
	DS-MT-4C	0.0023	1264.2624	5276.3989
	SM-MT-4C	0.0024	1166.2446	5018.9659

Tabela 6.1: Distribuição do tempo de execução.

A Figura 6.5, apresenta a distribuição dos tempos de processamento de cada fase de execução do *Benchmark*.

Observando o gráfico de distribuição do tempo de execução dos estágios, podemos observar que em média 78% do tempo de execução é consumido no estágio de processamento das imagens, e que 22% do tempo de execução é consumido no estágio de envio das imagens para o servidor.

Figura 6.5: Gráfico de distribuição do tempo de execução.

Na Figura 6.4, observamos que o tempo de *upload* e maior para operações *Single Threads* (ST), porque o armazenamento dos dados ocorre de forma sequencial nas diversas unidades de armazenamento CSD, não aproveitando o paralelismo existente.

A Figura 6.4, mostra também que o tempo de processamento e maior para operações *Multi Threads* (MT), quando simulamos 4 unidades de armazenamento CSD, devido a concorrência pelos recursos de memoria e processamento da maquina usada nos experimentos.

6.4 COMPARAÇÃO TODAS AS CONFIGURAÇÕES

Quando avaliamos o Tempo de Execução Total, considerando a execução de todos os estágios, MKDIR, UPLOAD e PROC, observamos que com o acréscimo de unidades CSD e o uso de *Shared Memory* (SM) para comunicação entre o servidor e os módulos de processamento, obtemos um ganho de aceleração (s*peedup*) de até 2,33 com 4 unidades CSD.

A simulação de 4 unidades CSD apresentou um desempenho baixo no estágio de processamento, PROC, que foi compensado pelo excelente resultado obtido no estágio de envio das imagens ao servidor, UPLOAD, Tabela 6.2.

#1. COMPARE ALL CONFIGURATIONS

	UPLOAD	SPEEDUP		PROC	SPEEDUP		TEMPO TOTAL	SPEEDUP
DS-MT-1C	3474.8353	0.9564	SM-ST-1C	12179.48989	0.9087	SM-ST-1C	15632.8541	0.9205
SM-ST-1C	3453.3635	0.9624	DS-MT-1C	11421.66055	0.9689	DS-MT-1C	14896.4967	0.9660
DS-ST-1C	3323.4447	1.0000	DS-ST-1C	11066.97516	1.0000	DS-ST-1C	14390.4208	1.0000
SM-MT-1C	3123.2359	1.0641	SM-MT-1C	10797.46826	1.0250	SM-MT-1C	13920.7050	1.0337
DS-ST-2C	2616.5962	1.2701	DS-ST-2C	7655.195835	1.4457	DS-ST-2C	10271.7929	1.4010
DS-MT-2C	2202.4704	1.5090	DS-MT-2C	7402.624804	1.4950	DS-MT-2C	9605.0964	1.4982
SM-ST-2C	2137.3474	1.5549	SM-MT-2C	6879.671804	1.6086	SM-ST-2C	9000.2360	1.5989
SM-MT-2C	1920.3308	1.7307	SM-ST-2C	6862.887451	1.6126	SM-MT-2C	8800.0032	1.6353
SM-ST-4C	1305.6174	2.5455	DS-MT-4C	5276.398878	2.0974	DS-MT-4C	6540.6636	2.2001
DS-MT-4C	1264.2624	2.6288	SM-ST-4C	5180.05565	2.1365	SM-ST-4C	6485.6756	2.2188
DS-ST-4C	1201.7843	2.7654	SM-MT-4C	5018.965917	2.2050	DS-ST-4C	6204.8780	2.3192
SM-MT-4C	1166.2446	2.8497	DS-ST-4C	5003.089617	2.2120	SM-MT-4C	6185.2129	2.3266

Tabela 6.2: Comparação do tempo de execução e da aceleração (*speedup*).

6.4.1 COMPARAÇÃO TODAS AS CONFIGURAÇÕES - UPLOAD

Analisando o estágio de envio das imagens ao servidor, UPLOAD, podemos verificar que este estágio obtém um ganho em aceleração (*speedup*) de até 2,85 com o aumento do paralelismo proporcionado pelo acréscimo de unidades de armazenamento CSD e com o uso de processamento *Multi Thread* (MT) nas operações de leitura e escrita dos arquivo.

Este estágio de execução, não obtém ganhos com o compartilhamento de memória usando *Shared Memory* (SM) entre o servidor e os módulos de processamento. Porque somente as operações de reconstrução dos arquivos e processamento usam o compartilhamento de memoria para otimizar o processamento. A Figura 6.6, apresenta a distribuição do tempo de execução e a aceleração (*speedup*) para a tarefa de UPLOAD dos dados.

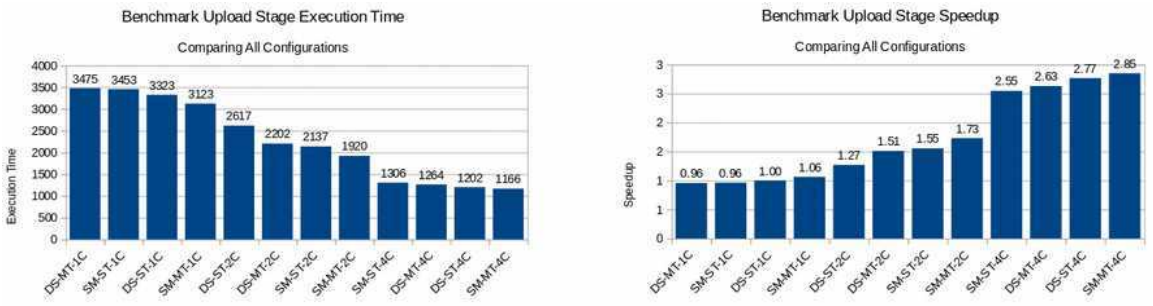

Figura 6.6: Distribuição do tempo de execução e de aceleração (*speedup*) na fase UPLOAD.

6.4.2 COMPARAÇÃO TODAS AS CONFIGURAÇÕES - PROC

O estágio de seleção, download e processamento das imagens, PROC, obtém um ganho em aceleração (*speedup*) de até 2,21 com o aumento do paralelismo através do acréscimo de unidades de armazenamento CSD, usando processamento *Multi Thread* (MT) nas operações de acesso ao disco, e armazenamento dos arquivos de imagem temporários em memória compartilhada durante o processamento.

O aumento no número de unidades de armazenamento CSD gerou um aumento na concorrência entre os processos por recursos de processamento e memória, e o melhor desempenho obtido com 4 unidades CSD é a configuração que efetua armazenamento em disco local, *Datastore* (DS), e uma linha de execução para acesso ao disco, *Single Thread* (ST), porque o modelo *Multi Threads* (MT) aumenta muito a concorrência por recursos da maquina usada para simulação. A Figura 6.7, apresenta a distribuição do tempo de execução e a aceleração (*speedup*) para a tarefa de PROC dos dados.

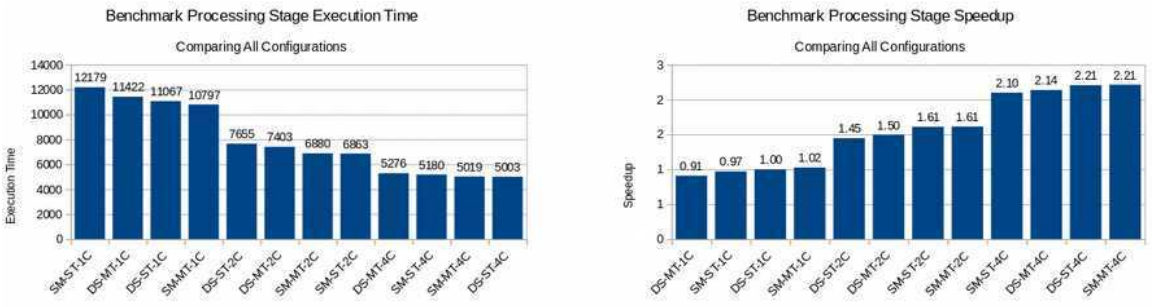

Figura 6.7: Distribuição do tempo de execução e aceleração (*speedup*) na fase PROC.

6.5 *DATASTORE* E *SHARED MEMORY*

O baixo resultado obtido com usando 4 unidades CSD ocorreu por falta de recursos no equipamento usado neste experimento, que possui somente 8 GB de Memoria RAM. Desta forma, com o acréscimo de processos, a concorrência pelo uso de memória se torna grande e o sistema operacional faz uso intenso da área de *swap* de disco, prejudicando o desempenho total do sistema, Tabela 6.3.

#2. COMPARE DATASTORE x SHMEM

	UPLOAD	SPEEDUP		PROC	SPEEDUP		TEMPO TOTAL	SPEEDUP
SM-ST-1C	3453.3635	0.9624	SM-ST-1C	12179.48989	0.9087	SM-ST-1C	15632.8541	0.9205
DS-MT-1C	3474.8353	0.9564	DS-MT-1C	11421.66055	0.9689	DS-MT-1C	14896.4967	0.9660
DS-ST-1C	3323.4447	1.0000	DS-ST-1C	11066.97516	1.0000	DS-ST-1C	14390.4208	1.0000
SM-MT-1C	3123.2359	1.0641	SM-MT-1C	10797.46826	1.0250	SM-MT-1C	13920.7050	1.0337
DS-ST-2C	2616.5962	1.0000	DS-ST-2C	7655.195835	1.0000	DS-ST-2C	10271.7929	1.0000
DS-MT-2C	2202.4704	1.1880	DS-MT-2C	7402.624804	1.0341	DS-MT-2C	9605.0964	1.0694
SM-ST-2C	2137.3474	1.2242	SM-ST-2C	6862.887451	1.1154	SM-ST-2C	9000.2360	1.1413
SM-MT-2C	1920.3308	1.3626	SM-MT-2C	6879.671804	1.1127	SM-MT-2C	8800.0032	1.1672
DS-MT-4C	1264.2624	0.9506	DS-MT-4C	5276.398878	0.9482	DS-MT-4C	6540.6636	0.9487
SM-ST-4C	1305.6174	0.9205	SM-ST-4C	5180.05565	0.9658	SM-ST-4C	6485.6756	0.9567
DS-ST-4C	1201.7843	1.0000	DS-ST-4C	5003.089617	1.0000	DS-ST-4C	6204.8780	1.0000
SM-MT-4C	1166.2446	1.0305	SM-MT-4C	5018.965917	0.9968	SM-MT-4C	6185.2129	1.0032

Tabela 6.3: Comparação do processamento com *Datastore* (DS) e com *Shared Memory* (SM).

Analisando os gráficos da Figura 6.8, observamos que o armazenamento dos arquivos temporários pode se beneficiar bastante do uso de memória compartilhada, *Shared Memory* (SM), entre o servidor e os módulos de processamento.

Entretanto, devido a capacidade limitada de memória do computador usado nos experimentos, o desempenho total do sistema em um ambiente simulado com 4 unidades CSD apresentou pouco ganho quando comparado com o armazenamento em disco local.

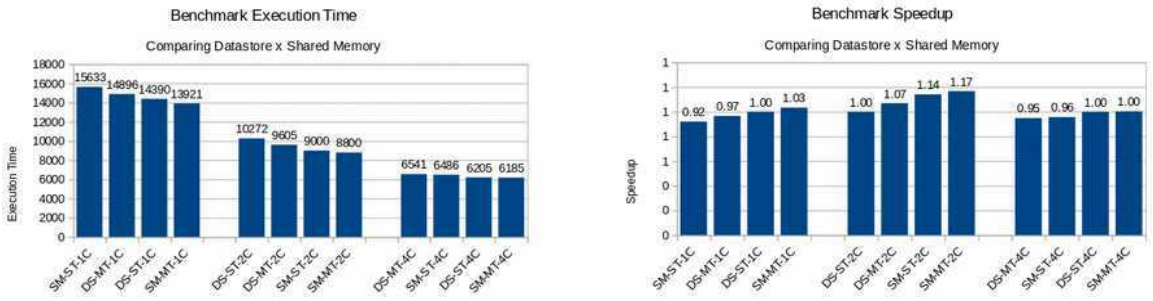

Figura 6.8: Gráfico de comparação do processamento com *Datastore* (DS) e com *Shared Memory* (SM).

6.5.1 *DATASTORE* E *SHARED MEMORY* - UPLOAD

Podemos observar nos gráficos da Figura 6.9, que quando comparamos o armazenamento dos arquivos temporários para processamento em disco local, *Datastore* (DS), com o compartilhamento de memória, *Shared Memory* (SM), a variação no ganho de *speedup* é pequena.

Isto ocorre, porque o estágio de UPLOAD não faz uso deste recurso, e a variação em *speedup* verificada no estágio de UPLOAD usando 2 unidades CSD, pode ser justificada pelo ganho proporcionado pelo uso de múltiplas linhas de execução para leitura e escrita no disco, *Multi Thread* (MT).

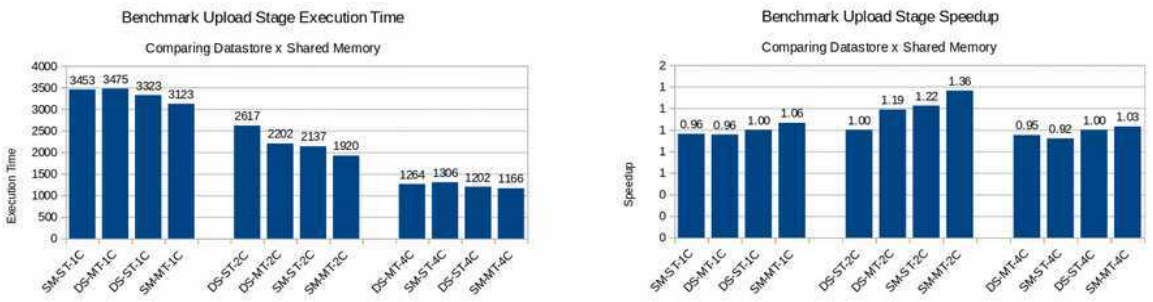

Figura 6.9: Gráfico de comparação da fase UPLOAD com *Datastore* (DS) e com *Shared Memory* (SM).

6.5.2 *DATASTORE* E *SHARED MEMORY* - PROC

No estágio de processamento das imagens, PROC, o armazenamento dos arquivos temporários pode se beneficiar bastante com o uso de memória compartilhada, *Shared Memory* (SM), entre o servidor e os módulos de processamento.

Analisando os gráficos da Figura 6.10, observamos que usando 2 unidades CSD obtemos ganhos de até 1,12 em *speedup* em relação a configuração de referência para o mesmo número de unidades CSD.

Usando 4 unidades CSD o desempenho é inferior a configuração de referência, porque a concorrência pelos recursos de processamento e memória do computador, com 8 GB RAM, usado neste experimento aumenta significativamente.

Usando somente 1 unidade CSD com o compartilhamento de arquivos em memória, *Shared Memory* (SM), e múltiplas linhas de leitura e escrita de dados, *Multi Threads* (MT), o ganho em *speedup* é de somente 1,03.

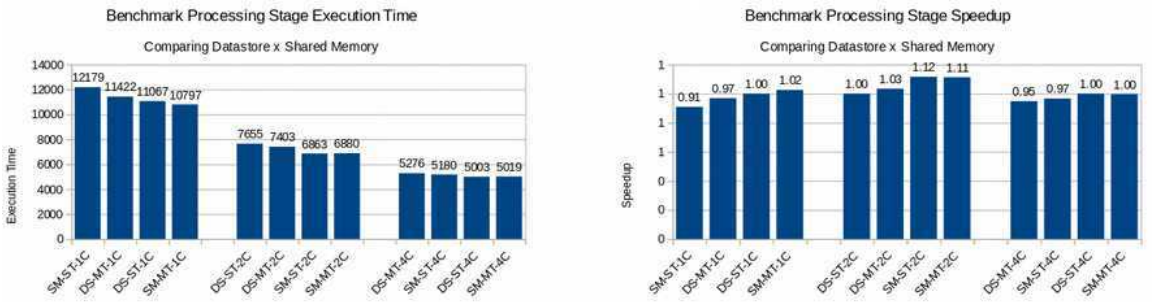

Figura 6.10: Gráfico de comparação da fase PROC com *Datastore* (DS) e com *Shared Memory* (SM).

6.6 *SINGLE THREAD* E *MULTI THREAD*

Analisando os resultados obtidos considerando os 3 estágios de execução do *Benchmark*: MKDIR, UPLOAD e PROC, com uma configuração com uma única linha de execução, *Single Thread* (ST), com o uso de múltiplas linhas de execução, *Multi Threads* (MT), verificamos que o uso de múltiplas linhas de execução para leitura e escrita em disco apresentou sempre um bom resultado quando comparamos o tempo total de execução do *Benchmark*.

O ganho de desempenho obtido usando 4 unidades CSDs, foi proporcionado pelo bom resultado do estágio de UPLOAD, que efetua o envio de imagens para o servidor. A Tabela 6.4, apresenta a distribuição do tempo de execução e a aceleração (*speedup*) para a tarefa de UPLOAD dos dados.

#3. COMPARE SINGLE THREAD x MULTI THREADS

	UPLOAD	SPEEDUP		PROC	SPEEDUP		TEMPO TOTAL	SPEEDUP
SM-ST-1C	3453.3635	0.9624	SM-ST-1C	12179.4899	0.9087	SM-ST-1C	15632.8541	0.9205
DS-ST-1C	3323.4447	1.0000	DS-ST-1C	11066.9752	1.0000	DS-ST-1C	14390.4208	1.0000
DS-ST-2C	2616.5962	1.2701	DS-ST-2C	7655.1958	1.4457	DS-ST-2C	10271.7929	1.4010
SM-ST-2C	2137.3474	1.5549	SM-ST-2C	6862.8875	1.6126	SM-ST-2C	9000.2360	1.5989
SM-ST-4C	1305.6174	2.5455	SM-ST-4C	5180.0557	2.1365	SM-ST-4C	6485.6756	2.2188
DS-ST-4C	1201.7843	2.7654	DS-ST-4C	5003.0896	2.2120	DS-ST-4C	6204.8780	2.3192
DS-MT-1C	3474.8353	1.0000	DS-MT-1C	11421.6605	1.0000	DS-MT-1C	14896.4967	1.0000
SM-MT-1C	3123.2359	1.1126	SM-MT-1C	10797.4683	1.0578	SM-MT-1C	13920.7050	1.0701
DS-MT-2C	2202.4704	1.5777	DS-MT-2C	7402.6248	1.5429	DS-MT-2C	9605.0964	1.5509
SM-MT-2C	1920.3308	1.8095	SM-MT-2C	6879.6718	1.6602	SM-MT-2C	8800.0032	1.6928
DS-MT-4C	1264.2624	2.7485	DS-MT-4C	5276.3989	2.1647	DS-MT-4C	6540.6636	2.2775
SM-MT-4C	1166.2446	2.9795	SM-MT-4C	5018.9659	2.2757	SM-MT-4C	6185.2129	2.4084

Tabela 6.4: Comparação do processamento com *Single Thread* (ST) e *Multi Thread* (MT).

6.6.1 *SINGLE THREAD* E *MULTI THREADS* - UPLOAD

Comparando a execução do estágio de UPLOAD, envio de imagens para o servidor, Figura 6.11, usando uma única linha de execução para leitura e escrita em disco, *Single Threads* (ST), com a utilização de múltiplas linhas de execução, *Multi Threads* (MT), verificamos a execução do *Benchmark* com múltiplas linhas de execução, *Multi Threads* (MT) apresentou sempre um bom resultado em relação ao uso de uma única linha de execução, *Single Thread* (ST), neste estágio.

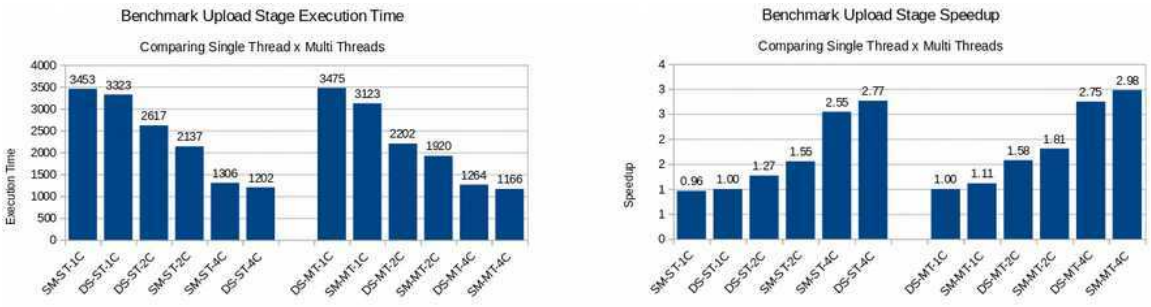

Figura 6.11: Gráfico de comparação na fase UPLOAD com *Single Thread* (ST) e com *Multi Thread* (MT).

6.6.2 *SINGLE THREAD* E *MULTI THREAD* - PROC

O uso múltiplas linhas de execução para leitura e escrita no disco, *Multi Threads* (MT), apresentou um bom resultado em relação ao uso de uma única linha de execução, *Single Threads* (ST), para o estágio de processamento das imagens do *Benchmark*, PROC, usando até 2 unidades CSD, Figura 6.12.

Para uma configuração do *Benchmark*, simulando 4 unidades CSD, o desempenho obtido foi melhor usando uma única linha de execução para leitura e escrita no disco, e os 4 processos simulando as unidades CSD.

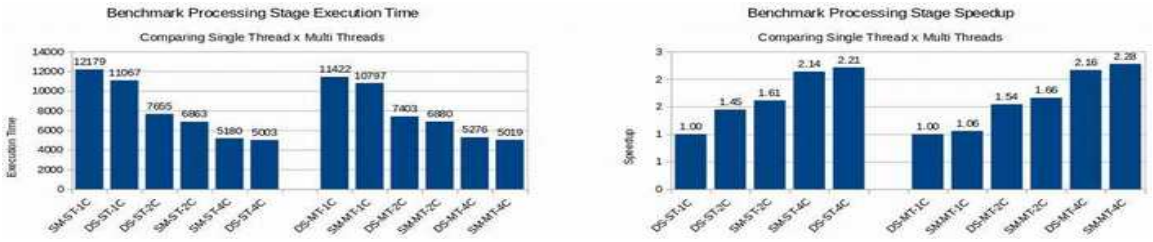

Figura 6.12: Gráfico de comparação na fase PROC com *Single Thread* (ST) e com *Multi Thread* (MT).

6.7 ESCALABILIDADE COM ACRÉSCIMO DE UNIDADES CSD

O acréscimo de unidades CSD e o aumento do paralelismo com a implementação de *Multi Threads* (MT) no acesso de leitura e escrita do disco, apresentou ganhos significativos de *speedup* que variaram de 1,57 à 1,63 usando 2 unidades CSD, e entre 2,68 à 2,75 usando 4 unidades CSD, Tabela 6.5.

O ganho em *speedup* usando 4 unidades CSD e com a implementação de leitura e escrita em disco usando *Single Thread* (ST), obteve desempenho superior a implementação *Multi Thread* (MT) devido a quantidade limitada de memória do computador usado neste experimento, que possui somente 8 GB.

#4. CHANGE THE NUMBER OF CSDs UNITS

	UPLOAD	SPEEDUP
DS-ST-1C	3323.4447	1.0000
DS-ST-2C	2616.5962	1.2701
DS-ST-4C	1201.7843	2.7654
SM-ST-1C	3453.3635	1.0000
SM-ST-2C	2137.3474	1.6157
SM-ST-4C	1305.6174	2.6450
DS-MT-1C	3474.8353	1.0000
DS-MT-2C	2202.4704	1.5777
DS-MT-4C	1264.2624	2.7485
SM-MT-1C	3123.2359	1.0000
SM-MT-2C	1920.3308	1.6264
SM-MT-4C	1166.2446	2.6780

	PROC	SPEEDUP
DS-ST-1C	11066.9752	1.0000
DS-ST-2C	7655.1958	1.4457
DS-ST-4C	5003.0896	2.2120
SM-ST-1C	12179.4899	1.0000
SM-ST-2C	6862.8875	1.7747
SM-ST-4C	5180.0557	2.3512
DS-MT-1C	11421.6605	1.0000
DS-MT-2C	7402.6248	1.5429
DS-MT-4C	5276.3989	2.1647
SM-MT-1C	10797.4683	1.0000
SM-MT-2C	6879.6718	1.5695
SM-MT-4C	5018.9659	2.1513

	TEMPO TOTAL	SPEEDUP
DS-ST-1C	14390.4208	1.0000
DS-ST-2C	10271.7929	1.4010
DS-ST-4C	6204.8780	2.3192
SM-ST-1C	15632.8541	1.0000
SM-ST-2C	9000.2360	1.7369
SM-ST-4C	6485.6756	2.4104
DS-MT-1C	14896.4967	1.0000
DS-MT-2C	9605.0964	1.5509
DS-MT-4C	6540.6636	2.2775
SM-MT-1C	13920.7050	1.0000
SM-MT-2C	8800.0032	1.5819
SM-MT-4C	6185.2129	2.2506

Tabela 6.5: Distribuição do tempo de execução por quantidade de unidades CSD.

O estágio de envio das imagens ao servidor, UPLOAD, obtém um ganho em *speedup* de até 2,85 com o aumento do paralelismo proporcionado pelo acréscimo de unidades CSD e com o uso de processamento *Multi Thread* (MT) nas operações de leitura e escrita dos arquivo.

O estágio de seleção, download e processamento das imagens, PROC, obtém um ganho em *speedup* de até 2,21 com o aumento do paralelismo através do acréscimo de unidades CSD, usando processamento *Multi Thread* (MT) nas operações de acesso ao disco, e armazenamento do arquivo de imagem temporário em memória compartilhada usando *Shared Memory* (SM).

O aumento no número de unidades CSD gerou um aumento na concorrência entre os processos por recursos de processamento e memória, e o melhor desempenho obtido com 4 unidades CSD é a configuração que efetua armazenamento em disco local, *Datastore* (DS), e uma linha de execução para acesso ao disco, *Single Thread* (ST).

Quando avaliamos o Tempo de Execução Total, considerando a execução de todos os estágios, MKDIR, UPLOAD e PROC, observamos que com o acréscimo de unidades CSD e o uso de *Shared Memory* (SM) para comunicação entre o servidor e os módulos de processamento, obtemos um ganho de *speedup* de até 2,33 usando 4 unidades CSD.

A simulação de 4 unidades CSD apresentou um baixo desempenho no estágio de processamento, PROC, que foi compensado pelo excelente resultado obtido no estágio de envio das imagens ao servidor, UPLOAD.

No estágio de processamento das imagens, PROC, o armazenamento dos arquivos temporários pode se beneficiar bastante com o uso de memória compartilhada, *Shared Memory* (SM), entre o servidor e os módulos de processamento.

Observamos que usando 2 unidades CSD obtemos ganhos de até 1,12 em aceleração (*speedup*) em relação a configuração de referência para o mesmo número de unidades CSD.

O baixo resultado obtido usando 4 unidades CSD ocorreu por falta de recursos no equipamento usado neste experimento com 8 GB de Memoria RAM).

O acréscimo de unidades CSD e o aumento do paralelismo com a implementação de *Multi Threads* (MT) no acesso de leitura e escrita do disco, apresentou ganhos significativos de *speedup* que variaram de 1,57 à 1,63 usando 2 unidades CSD, e entre 2,68 à 2,75 usando 4 unidades CSD.

6.8 ANALISE DE DESEMPENHO *(PROFILE)*

iv) Quarto Experimento (Apos Otimização - PROFILE)

Nesta nova fase dos experimentos analisamos as rotinas de maior custo e que sao mais executadas, e em seguida realizamos simplificações e otimizações. A Tabela 6.6, apresenta o resultado do *profile* obtido da execução do sistema.

Code	Function Name	% Execution Time	Cumulative Time (s)	Self Time (s)	# Self Calls	Total Time per Call (s)	Time per Call
1	memNCpyUtil(unsigned char*^unsigned char*^int)	49.55	16.87	16.87	66988	0	0
2	memSetNullUtil(unsigned char*^int)	37.45	29.62	12.75	44410	0	0
3	CDskSuperBlockMan::findFirstFreeSuperBlock(long^	8.02	32.35	2.73	22144	0	0
4	CDskBlockMan::findBlockByBlockNum(long^long^dsk	2.76	33.29	0.94	22226	0	0
5	CDskBlockMan::findFirstFreeBlock(long^long^dsk_bl	2.06	33.99	0.7	22185	0	0
6	getHash(char*)	0.09	34.02	0.03	44238	0	0
7	CSequence::debugEntry(int^char const*^char const*^s	0.03	34.03	0.01	243725	0	0
8	CDskSuperBlockMan::findAllByPathOidAndDiskDev(l	0.03	34.04	0.01	15	0	0.37
9	CDskSuperBlockMan::countItemsByPathOidAndDiskD	0.03	34.05	0.01	15	0	0
10	warnMsg(int^char const*^char const*^char const*)	0	34.05	0	620905	0	0
11	getCurrentTimestamp()	0	34.05	0	66487	0	0
12	strNCmpCaseUtil(char*^char*^int^int)	0	34.05	0	53390	0	0
13	strNCpyUtil(char*^char*^int)	0	34.05	0	44997	0	0
14	openFileUtil(_IO_FILE**^char const*^char const*^long)	0	34.05	0	44445	0	0
15	strNCmpUtil(char*^char*^int)	0	34.05	0	44424	0	0
16	CAppMain::getSequencePtr()	0	34.05	0	44341	0	0
17	CSequence::nextVal(char*)	0	34.05	0	44303	0	0
18	CSequence::findItem(char*^sequence_struct**)	0	34.05	0	44281	0	0
19	CDskDiskDev::readBlock(long^long^long^unsigned cl	0	34.05	0	22200	0	0
20	CDskDiskDev::getOid()	0	34.05	0	22166	0	0
21	CDskDiskGroup::getOid()	0	34.05	0	22159	0	0
22	CDskDiskDev::writeBlock(long^long^long^unsigned c	0	34.05	0	22138	0	0
23	CDataTable::checkScoreList(double*^long^double)	0	34.05	0	8160	0	0
24	CCfgConfigTag::debugEntry(int^char const*^char const	0	34.05	0	1038	0	0
25	strSetEmptyUtil(char*)	0	34.05	0	993	0	0
26	strNCpyCaseUtil(char*^char*^int^int)	0	34.05	0	926	0	0
27	CCfgConfigTag::addNewConfigTag(int^int^char*)	0	34.05	0	926	0	0
28	CDskDiskManExec::getThreadAtPtr(long)	0	34.05	0	780	0	0
29	CDataTable::debugEntry(int^char const*^char const*^d	0	34.05	0	390	0	0
30	CAppExec::addDataHist(double)	0	34.05	0	390	0	0
31	CDskDiskManExec::resetThread(dsk_diskman_thread_	0	34.05	0	174	0	0
32	strRemoveEolUtil(char*^int)	0	34.05	0	139	0	0
33	errMsgIfNull(char const*^char const*^char const*^void	0	34.05	0	132	0	0
34	CCfgConfigTag::setConfigTagValue(char*^char*)	0	34.05	0	117	0	0
35	CCfgConfigTag::findConfigTagByName(char*)	0	34.05	0	117	0	0
36	strNCatUtil(char*^char*^int)	0	34.05	0	116	0	0

Tabela 6.6: Resultado da analise do *profile* da execução do sistema.

A Tabela 6.7, apresenta as rotinas que consomem o maior tempo de processamento, os principais gargalos no processamento do sistema.

Code	Function Name	% Execution Time	Cumulative Time (s)	Self Time (s)	# Self Calls	Total Time per Call (s)	Time per Call
3	CDskSuperBlockMan::findFirstFreeSuperBlock(long^l	8.02	32.35	2.73	22144	0	0
4	CDskBlockMan::findBlockByBlockNum(long^long^dsk	2.76	33.29	0.94	22226	0	0
5	CDskBlockMan::findFirstFreeBlock(long^long^dsk_bl	2.06	33.99	0.7	22185	0	0
6	getHash(char*)	0.09	34.02	0.03	44238	0	0
8	CDskSuperBlockMan::findAllByPathOidAndDiskDev(l	0.03	34.04	0.01	15	0	0.37
9	CDskSuperBlockMan::countItemsByPathOidAndDiskD	0.03	34.05	0.01	15	0	0
19	CDskDiskDev::readBlock(long^long^long^unsigned ch	0	34.05	0	22200	0	0
22	CDskDiskDev::writeBlock(long^long^long^unsigned c	0	34.05	0	22138	0	0

Tabela 6.7: Lista com as rotinas que consomem o maior tempo de processamento.

As rotinas que consomem o maior tempo de processamento podem ser vistas nas Figuras 6.13 e 6.14. Estas rotinas devem ser otimizadas para melhorar o desempenho total do sistema.

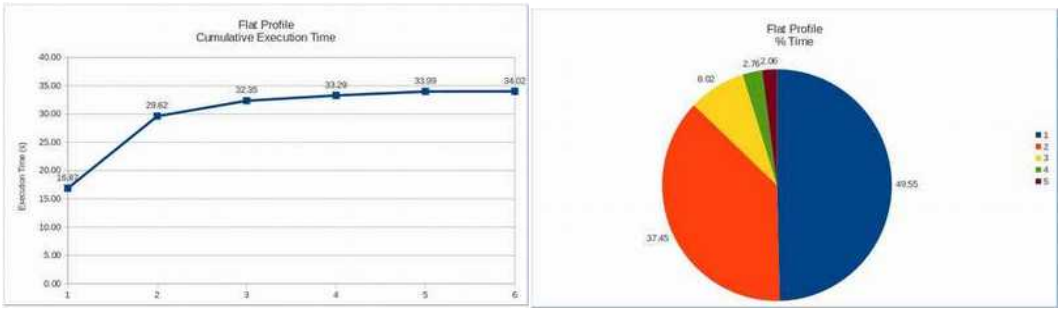

Figura 6.13: Distribuição das rotinas que consomem o maior tempo de processamento.

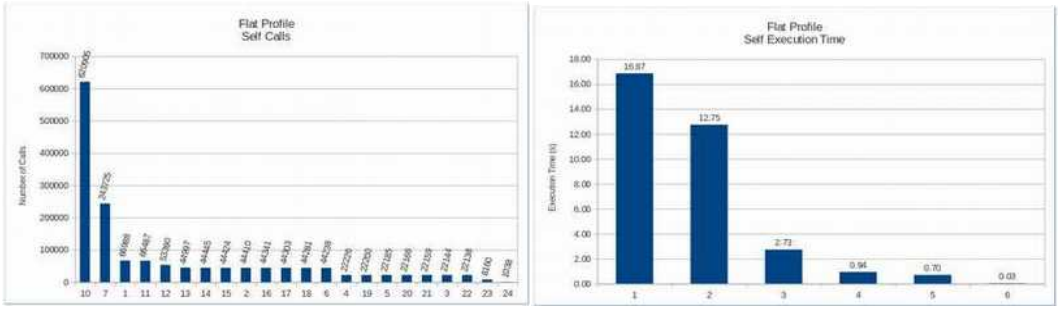

Figura 6.14: Distribuição das rotinas que consomem o maior tempo de processamento.

Das Figuras 6.12 e 6.13, podemos observar que os maiores gargalos no processamento se resumem as 7 rotinas listadas abaixo, e que a otimização delas resulta em uma melhora significativa no processamento do sistema.

* CdskSuperBlockMan::findFirstFreeSuperBlock();
* CdskSuperBlockMan::findAllByPathOidAndDiskDev();
* CdskSuperBlockMan::countItemsByPathOidAndDiskDev();
* CdskBlockMan::findFirstFreeBlock();
* CdskBlockMan::findBlockByBlockNum();
* CdskDiskDev::readBlock()
* CdskDiskDev::writeBlock()

6.8.1 OTIMIZACOES REALIZADAS

Para otimizar o sistema HORUS Image Server, optei pelas seguintes medidas:

i) Adição de índices nas tabelas de dados.

ii) Implementação da estrutura de diretório, da lista de Superblocos e da lista de Blocos armazenados em memória usando *Hashtables*.

iii) Aumentar o número de *threads* simultaneamente disparadas, com o objetivo de melhorar o desempenho no processamento das solicitações de leitura ou gravação de dados no disco.

6.8.2 RESULTADOS APOS OTIMIZAÇÃO

O processo de otimização foi quebrado em duas etapas. Na primeira, tratamos de um problema simples de identificação de blocos vazios para gravação dos dados, e na segunda, da inclusão de estruturas otimizadas para localizar rapidamente a sequencia de blocos de um mesmo arquivo.

(a) Otimização Efetuada – PARTE 1

i) Criação de Classes de estrutura de dados para auxiliar a implementação dos índices das tabelas do sistema: FIFO; STACK; DOUBLE-LINKED LIST; HASHTABLE; BALANCED TREE (RUBRO-NEGRA);

ii) Otimização das funções para evitar a busca por blocos livres no caso de haver espaço disponível na tabela.

* CDskSuperBlockMan::findFirstFreeSuperBlock();
* CDskBlockMan::findFirstFreeBlock();

(b) Otimização Efetuada – PARTE 2

i) Incluir a quantidade de superblocos e blocos de um arquivo armazenados em cada unidade CSD.

ii) Implementação da estrutura de Diretório, da lista de Superblocos e da lista de Blocos armazenados em memória usando “Hashtables”.

iii) Implementação de índice de superblocos e blocos usando “Hashtables” e “Double-Linked-List”, com chaves “path_oid” e “block_num”.

Quantidade de superblocos e blocos obtida do índice correspondente

* CdskSuperBlockMan::countItemsByPathOidAndDiskDev();
* CdskSuperBlockMan::findAllByPathOidAndDiskDev();
* CdskBlockMan::findBlockByBlockNum();

Aumentar o número de *threads* simultaneamente disparadas, com o objetivo de melhorar o desempenho no processamento das solicitações de leitura ou gravação de dados no disco.

Após otimização das estruturas de superblocos e blocos não foi necessário efetuar otimização na leitura e escrita de blocos

* CdskDiskDev::readBlock()
* CdskDiskDev::writeBlock()

Apos as otimizações. Observamos na Tabela 6.8, a melhora no processamento do sistema com os itens otimizados destacados em verde.

Code	Function Name	% Execution Time	Cumulative Time (s)	Self Time (s)	# Self Calls	Total Time per Call (s)	Time per Call
1	memNCpyUtil(unsigned char**unsigned char**int)	49.55	16.87	16.87	66988	0	0
2	memSetNullUtil(unsigned char**int)	37.45	29.62	12.75	44410	0	0
3	CDskSuperBlockMan::findFirstFreeSuperBlock(long^long^long^long^long^long^dsk_superblock_struct**^long*)	8.02	32.35	2.73	22144	0	0
4	CDskBlockMan::findBlockByBlockNum(long^long^dsk_block_struct**^long*^long*)	2.76	33.29	0.94	22226	0	0
5	CDskBlockMan::findFirstFreeBlock(long^long^dsk_block_struct**^long*^long*)	2.06	33.99	0.7	22185	0	0
6	getHash(char*)	0.09	34.02	0.03	44238	0	0
7	CSequence::debugEntry(int*char const*^char const*^sequence_struct*)	0.03	34.03	0.01	243725	0	0
8	CDskSuperBlockMan::findAllByPathOidAndDiskDev(long^long^dsk_superblock_struct**^long*)	0.03	34.04	0.01	15	0	0.37
9	CDskSuperBlockMan::countItemsByPathOidAndDiskDev(long^long^long*)	0.03	34.05	0.01	15	0	0
10	warnMsg(int*char const*^char const*^char const*)	0	34.05	0	620905	0	0
11	getCurrentTimestamp()	0	34.05	0	66487	0	0
12	strNCmpCaseUtil(char*^char*^int*int)	0	34.05	0	53390	0	0
13	strNCpyUtil(char*^char*^int)	0	34.05	0	44997	0	0
14	openFileUtil(_IO_FILE**^char const*^char const*^long)	0	34.05	0	44445	0	0
15	strNCmpUtil(char*^char*^int)	0	34.05	0	44424	0	0
16	CAppMain::getSequencePtr()	0	34.05	0	44341	0	0
17	CSequence::nextVal(char*)	0	34.05	0	44303	0	0
18	CSequence::findItem(char**sequence_struct**)	0	34.05	0	44281	0	0
19	CDskDiskDev::readBlock(long^long^long^unsigned char**dsk_block_struct**^long*^long*)	0	34.05	0	22200	0	0
20	CDskDiskDev::getOid()	0	34.05	0	22166	0	0
21	CDskDiskGroup::getOid()	0	34.05	0	22159	0	0
22	CDskDiskDev::writeBlock(long^long^long^unsigned char**dsk_block_struct**^long*^long*)	0	34.05	0	22138	0	0
23	CDataTable::checkScoreList(double**long*double)	0	34.05	0	8160	0	0
24	CCfgConfigTag::debugEntry(int*char const*^char const*^config_tag_struct*)	0	34.05	0	1038	0	0
25	strSetEmptyUtil(char*)	0	34.05	0	993	0	0
26	strNCpyCaseUtil(char*^char*^int*int)	0	34.05	0	926	0	0

Tabela 6.8: Resultado da analise do *profile apos a* otimização do sistema.

6.8.3 COMPARAÇÃO CONFIGURAÇÕES APOS OTIMIZAÇÃO

A versão final que foi otimizada ao máximo foi o servidor HORUS Image Server v2.6.20220417. Nesta versão resolvemos todos os 7 problemas de desempenhos identificados no *profile*, e analisamos o processamento dos dados usando leitura e escrita *Multi Thread* (MT) *e Shared Memory* (SM) para processamento das imagens.

Nesta versão simulamos somente o uso de 4 unidades de armazenamento CSD. Na Tabela 6.9, esta versão esta identificada por SM-MT-4C-O (s*hared memory*, m*ulti thread*, 4 unidades CSD, e Otimizada).

#1. COMPARE ALL CONFIGURATIONS

	UPLOAD	SPEEDUP
DS-MT-1C	3474.8353	0.9564
SM-ST-1C	3453.3635	0.9624
DS-ST-1C	3323.4447	1.0000
SM-MT-1C	3123.2359	1.0641
DS-ST-2C	2616.5962	1.2701
DS-MT-2C	2202.4704	1.5090
SM-ST-2C	2137.3474	1.5549
SM-MT-2C	1920.3308	1.7307
SM-ST-4C	1305.6174	2.5455
DS-MT-4C	1264.2624	2.6288
DS-ST-4C	1201.7843	2.7654
SM-MT-4C	1166.2446	2.8497
SM-MT-4C-O	531.7809	6.2497

	PROC	SPEEDUP
SM-ST-1C	12179.48989	0.9087
DS-MT-1C	11421.66055	0.9689
DS-ST-1C	11066.97516	1.0000
SM-MT-1C	10797.46826	1.0250
DS-ST-2C	7655.195835	1.4457
DS-MT-2C	7402.624804	1.4950
SM-MT-2C	6879.671804	1.6086
SM-ST-2C	6862.887451	1.6126
DS-MT-4C	5276.398878	2.0974
SM-ST-4C	5180.05565	2.1365
SM-MT-4C	5018.965917	2.2050
DS-ST-4C	5003.089617	2.2120
SM-MT-4C-O	6188.274868	1.7884

	TEMPO TOTAL	SPEEDUP
SM-ST-1C	15632.8541	0.9205
DS-MT-1C	14896.4967	0.9660
DS-ST-1C	14390.4208	1.0000
SM-MT-1C	13920.7050	1.0337
DS-ST-2C	10271.7929	1.4010
DS-MT-2C	9605.0964	1.4982
SM-ST-2C	9000.2360	1.5989
SM-MT-2C	8800.0032	1.6353
DS-MT-4C	6540.6636	2.2001
SM-ST-4C	6485.6756	2.2188
DS-ST-4C	6204.8780	2.3192
SM-MT-4C	6185.2129	2.3266
SM-MT-4C-O	6720.0582	2.1414

Tabela 6.9: Comparação do tempo de execução e da aceleração (speedup) apos otimização.

6.8.3.1 COMPARAÇÃO CONFIGURAÇÕES - UPLOAD

Na Figura 6.15, podemos analisar o estágio de envio das imagens ao servidor, UPLOAD, e verificar que a otimização gerou um acréscimo em aceleração (*speedup*) de 6,25 neste estágio. Este gigantesco ganho de desempenho ocorreu porque a otimização do sistema foi focalizada nas rotinas relacionadas a leitura e gravação dos dados.

* CDskSuperBlockMan::findFirstFreeSuperBlock();
* CDskBlockMan::findFirstFreeBlock();
* CdskSuperBlockMan::countItemsByPathOidAndDiskDev();
* CdskSuperBlockMan::findAllByPathOidAndDiskDev();
* CdskBlockMan::findBlockByBlockNum();

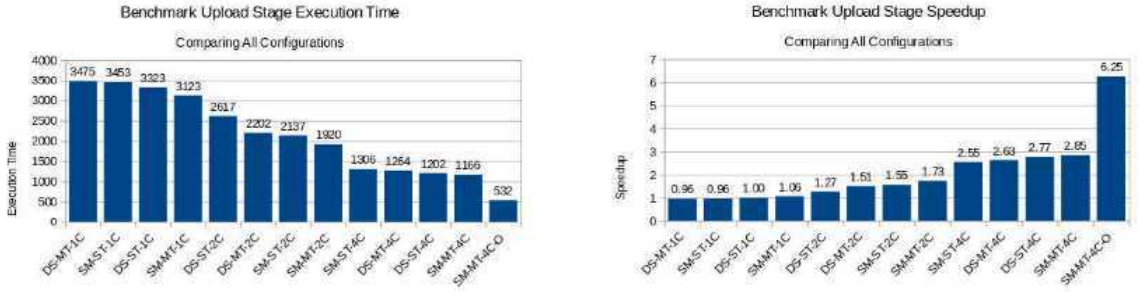

Figura 6.15: Distribuição do tempo de execução e de aceleração (speedup) na fase UPLOAD.

6.8.3.2 COMPARAÇÃO CONFICURAÇÕES - PROC

Na Figura 6.16, podemos verificar que o estágio de processamento das imagens não teve o mesmo desempenho com as otimizações realizadas, e ficou com desempenho comparável ao processamento com 2 unidades de armazenamento CSD. Isto e, com acréscimo em aceleração (*speedup*) de 1,79.

O baixo desempenho foi provocado pelo recurso disponível para processamento no ambiente de simulação, e deve ser bem melhor no processamento nas unidades de armazenamento CSD fornecidas pela NGD Systems.

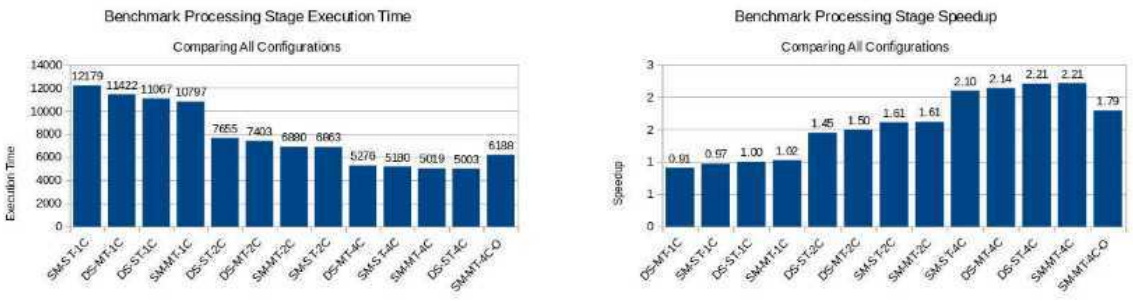

Figura 6.16: Distribuição do tempo de execução e de aceleração (speedup) na fase PROC.

6.6 GANHO POTENCIAL EM AMBIENTE REAL (UNIDADES CSD)

Os experimentos foram realizados em um 1 Notebook SAMSUNG, Modelo NP550PSC-AD1BR, com Processador Intel Core i7-3630QM de 2,40 GHz, com 4 Núcleos de Processamento e suporte para até 8 *Threads*, 8 GB de Memória RAM, e disco rígido convencional de HDD 1.0 TB.

Cada imagem do *Benchmark* possui mais de 400 MB de espaço em disco, e cerca de 2 GB de espaço em memoria. Desta forma, o acréscimo de unidades de armazenamento CSD nas simulações provoca conflitos entre os processos que concorrem por recursos de processamento.

Certamente, o uso das unidades de armazenamento CSD, ira distribuir os processos pelos equipamentos e evitar a concorrência por recursos do servidor hospedeiro.

6.7 PROPOSTAS PARA OTIMIZACAO DO MODULO FPGA

O modulo FPGA presente nas unidades de armazenamento Newport CSD, da empresa NGD Systems, não possui boa documentação.

Porem, podemos supor que em ambientes dedicados ao processamento de imagens, como nos servidores de imagens que atendem ao serviço do Censo Demográfico realizado pelo IBGE, podemos incluir nestes módulos chamadas de sistemas (*syscall*), capazes de otimizar parte do processamento e do tratamento das imagens, como a compactação das imagens, a projeção de coordenadas, além de funções de busca.

No artigo "*POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database*" [14], este modulo foi usado para otimizar as comparações de cadeias de caracteres (*strings*).

Capítulo 7

Conclusões e Trabalhos Futuros

Neste trabalho conduzimos os experimentos de forma simulada, sem usar as unidades de armazenamento Newport CSD da empresa NGD Systems, e utilizamos 1 Notebook SAMSUNG, Modelo NP550PSC-AD1BR, com Processador Intel Core i7-3630QM de 2,40 GHz, com 4 Núcleos de Processamento e suporte para até 8 Threads, 8 GB de Memória RAM, e disco rígido convencional de HDD 1.0 TB.

Para reduzir o tempo de processamento dos experimentos no ambiente simulado, trabalhamos com a consulta TS-1, que recupera 4,7% dos dados, recuperando 190 imagens para processamento [14].

Neste *Benchmark* realizamos as operações mais frequentes em sistemas de geoprocessamento, que são as operações de conversão de formato de imagens, projeção das imagens no sistema de coordenadas UTM, e a simplificação do tamanho das imagens para visualização através da Internet [19].

As imagens selecionadas para processamento possuem mais de 400 MB de espaço em disco, e cerca de 2 GB de espaço em memoria. Desta forma, a utilização de unidades de armazenamento CSD irá melhorar de forma significativa o desempenho geral do sistema, distribuindo os processos pelos equipamentos e evitando a concorrência por recursos do servidor hospedeiro.

Este trabalho foi realizado em três etapas, e foi sendo melhorado continuamente até obtermos a sua versão final com armazenamento de dados distribuídos a nível de bloco e com otimização das rotinas, o servidor HORUS Image Server v2.6.20220417.

i) Primeiro Experimento (Experimento Inicial)

No primeiro experimento os dados eram distribuídos pelas unidades de armazenamento CSD, no nível de arquivos de imagens, a leitura e escrita dos dados era sequencial e o aumento do paralelismo ocorria com o acréscimo de unidades de armazenamento CSD durante a simulação.

Desta forma, o melhor desempenho foi obtido com a configuração que simulou 4 unidades de armazenamento CSD, onde obtivemos ganhos de aceleração (*speedup*) de 1,69 na conversão do formato das imagens, de 1,20 na projeção das imagens do sistema de coordenadas geográficas para UTM, e de 1,48 na simplificação das imagens para ¼ do tamamnho.

ii) Segundo Experimento

No segundo experimento, com a versão do sistema HORUS Image Server v2.2.20220114, os dados eram distribuídos pelas unidades de armazenamento CSD, no nível de blocos, e a leitura e escrita dos dados era processada por múltiplas linhas de execução.

Além disso, as unidades de armazenamento CSD, montavam os arquivos a partir dos blocos distribuídos pelas unidades, e armazenava em memória compartilhada usando *shared memory*, permitindo que o processamento de conversão, projeção e simplificação das imagens fossem realizados em memória.

Desta forma, o melhor desempenho foi obtido com a configuração que simulou 4 unidades de armazenamento CSD, usando armazenamento compartilhado dos dados para processamento, *Shared Memory* (SM), e leitura e escrita dos dados *Multi Thread* (MT), onde obtivemos ganhos de aceleração (*speedup*) de 2,33 comparendo todas as etapas do processamento.

iii) Terceiro Experimento (Otimização)

No terceiro experimento, com a versão do sistema HORUS Image Server v2.6.20220417, os dados eram distribuídos pelas unidades de armazenamento CSD, no nível de blocos, a leitura e escrita dos dados era processada por múltiplas linhas de execução, foi implementado memória compartilhada, *shared* memory, para montagem e processamento dos arquivos, e foi realizada a otimização de algumas rotinas que geravam gargalos na execução do *Benchmark*.

* CdskSuperBlockMan::findFirstFreeSuperBlock();
* CdskSuperBlockMan::findAllByPathOidAndDiskDev();
* CdskSuperBlockMan::countItemsByPathOidAndDiskDev();
* CdskBlockMan::findFirstFreeBlock();
* CdskBlockMan::findBlockByBlockNum();
* CdskDiskDev::readBlock()
* CdskDiskDev::writeBlock()

Desta forma, o desempenho obtido com a configuração que simulou 4 unidades de armazenamento CSD, usando armazenamento compartilhado dos dados para processamento, *Shared Memory* (SM), leitura e escrita dos dados *Multi Thread* (MT), e otimização das rotinas, obteve ganhos de aceleração (*speedup*) de 2,14 comparando todas as etapas do processamento.

Este desempenho foi inferior ao desempenho registrado pelo sistema sem aceleração que teve ganhos de aceleração (*speedup*) de 2,33. Isto se explica pelo maior nível de concorrência por recursos do sistema durante a execução simulada, mas deverá ser muito melhor em um ambiente com as unidades de armazenamento Newport CSD.

Embora o desempenho geral da versão do sistema HORUS Image Server v2.6.20220417, tenha sido pior que a versão anterior do sistema, o seu resultado na etapa de UPLOAD dos dados foi de 6,25 em aceleração (*speedup*), porque nesta etapa não tem processamento das imagens e os recursos presentes na maquina hospedeira foram mais que suficientes para execução desta tarefa.

7.1 TRABALHOS FUTUROS

Pensando nos futuros trabalhos de pesquisa na área de *computational storage devices* (CSD), podemos prosseguir com os estudos em três linhas.

A primeira, é executar o sistema em um ambiente de produção com equipamentos reais Newport CSD fornecidos pela NGD Systems [70], ou os SmartSSD de fabricação da Xilinx/Samsung [71].

Este trabalho seria realizado preferencialmente em um projeto de boa visibilidade como o Censo Demográfico do IBGE, onde poderemos avaliar de forma eficiente o desempenho do sistema e confirmar as vantagens das unidades de armazenamento com processamento embutido (CSD).

A segunda, explora o modulo FPGA presente nas unidades de armazenamento Newport CSD. Nestes módulos podemos incluir chamadas de sistema (*syscall*) dedicadas ao processamento de imagens, e capazes de otimizar parte do processamento e do tratamento das imagens.

A terceira, é aprimorar o sistema operacional Linux Ubuntu 16.04.5 LTS [66] usado nas unidades Newport CSD, com um sistema operacional baseado em Linux e dedicado ao sistema de armazenamento e processamento eficiente destas unidades.

Este tipo de solução é bastante comum, e pode ser comparado as soluções de virtualização como o Citrix Xen Server [67] ou do Oracle VM Virtual Box [68 e 69], que fornecem uma camada de serviços específicos para virtualização sobre o ambiente Linux.

De forma similar, podemos incluir serviços específicos para otimização dos dados de imagens, bancos de dados relacionais, criptografia e outros, usando um ambiente próprio para unidades de armazenamento com processamento embutido.

Apêndice A

O Servidor de Imagens – HORUS

Para realização das análises foi desenvolvido um servidor de banco de dados distribuído, HORUS Image Server v2.6.20220417, que possui capacidade de armazenar os dados de forma distribuída entre as unidades de armazenamento CSD, usando blocos de 4 KB, que é mais eficientes para transporte através do barramento PCI-e usando protocolo NVM-e.

Além disso, o processamento *multi thread* para leitura e gravação dos dados em disco, e o uso de memória compartilhada, *shared memory*, para otimizar o processamento em memória das imagens pelos módulos de conversão, projeção e simplificação.

O objetivo principal do servidor HORUS Image Server v2.6.20220417, que é prover uma infraestrutura capaz de receber os dados coletados por sensores e câmeras de vigilância espalhadas pela cidade, e identificadas por E1, E2, …, E8, na Figura A.1, pré-selecionar os dados nos equipamentos remotos, *edge computing*, com objetivo de reduzir o volume de dados entre os equipamentos remotos e o servidor.

Em seguida, enviar os dados através da Internet para o *data center*, onde o dado será armazenado de forma distribuída e com redundância. Em seguida, os dados serão classificado usando Inteligência Computacional, permitindo a consulta rápida sobre a informação coletada, Figura A.1.

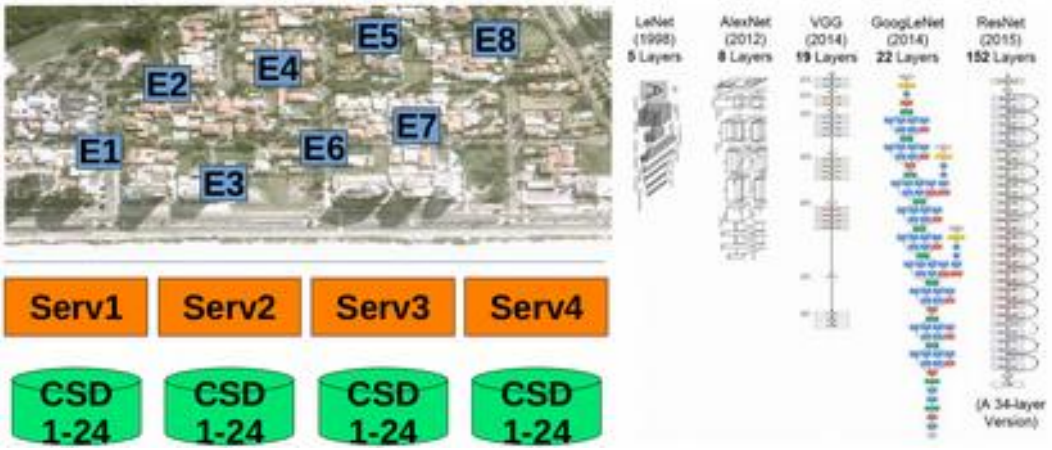

Figura A.1: Objetivo da implementação do servidor HORUS Image Server v2.6.20220417.

O processamento na borda será realizado usando TinyML, que simplifica o modelo de Inteligência Computacional e possibilita a filtragem de grande parte dos dados coletados.

Outro servidor de imagens usa um novo conceito de aprendizados, denominado *Federated Learning*, onde o servidor recebe os dados dos equipamentos remotos e periodicamente aprimora o modelo de inteligência computacional e retransmite o novo modelo para as unidades remotas, Figura A.1.

A.1 HORUSREMOTE – CLIENT ACCESS DRIVE

A Figura A.2, apresenta o fluxo de dados do controlador de acesso ao servidor de imagens, HORUS Image Server v2.6.20220417. Neste controlador existe uma tarefa que envia as solicitações da aplicação cliente e uma tarefa que fica ouvindo um canal de retorno dos dados. O controlador de acesso ao servidor foi desenvolvido nas linguagens de programação Java e Python, permitindo que outras aplicações interajam com o servidor de imagens.

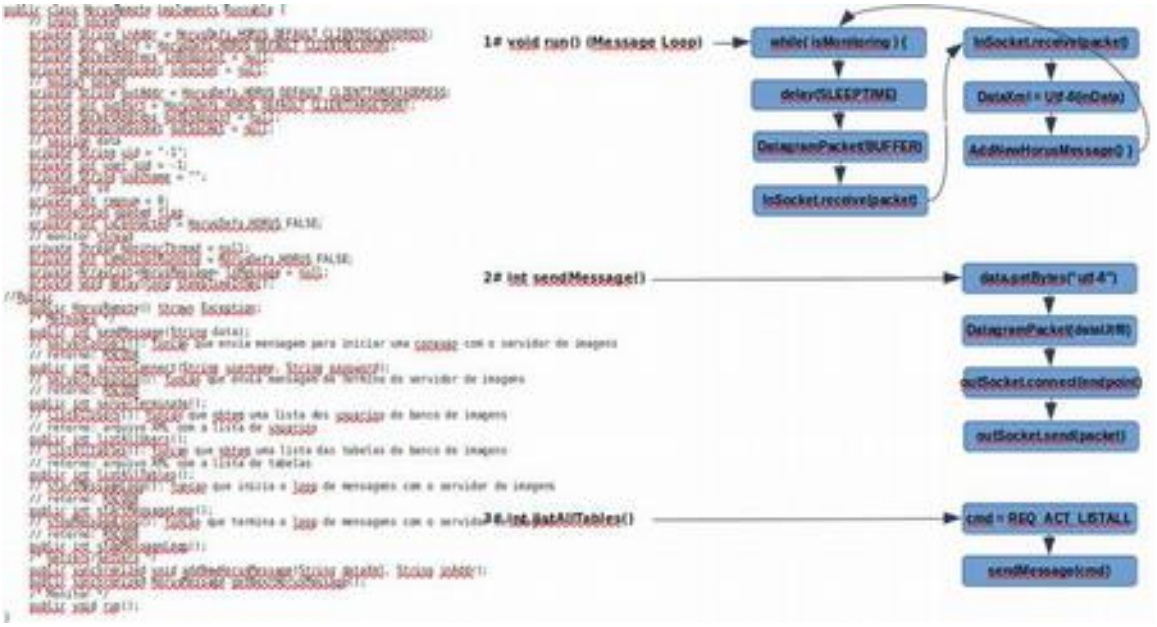

Figura A.2: HORUSREMOTE – CLIENT ACCESS DRIVE

A.2 HORUSSERVER

O servidor de imagens HORUS Image Server v2.6.20220417, possui dois módulos distintos. O modulo servidor, HORUSSERVER, que executa no servidor hospedeiro e interage com as chamadas do controlador de acesso do cliente, e o modulo de trabalho, HORUSWORKER, que executa nas unidades de armazenamento CSD.

A.2.1 HORUSSERVER – SERVER LISTENER

A Figura A.3, apresenta o fluxo de dados da tarefa de monitoramento do canal de comunicação, *Server Listener*, do servidor de imagens HORUS Image Server v2.6.20220417. Esta tarefa fica aguardando a chegada de solicitações dos clientes e quando recebe uma solicitação, dispara uma tarefa para execução.

Figura A.3: HORUSSERVER – SERVER LISTENER

A.2.2 HORUSSERVER – COMMAND INTERPRETER

A Figura A.4, apresenta o fluxo de dados da tarefa do interpretador de comandos, *Command Interpreter*, do servidor de imagens HORUS Image Server v2.6.20220417. Esta tarefa recebe e efetua um *parser* sobre os dados recebidos, interpretando o comando.

Esta é uma tarefa auxiliar do monitor do canal de comunicação e serve para distribuir de forma adequada as solicitações para as tarefas responsáveis pela execução.

Figura A.4: HORUSSERVER – COMMAND INTERPRETER

A.2.3 HORUSSERVER – COMMAND EXECUTER

A Figura A.5, apresenta o fluxo de dados da tarefa de execução de comandos, *Command Executer*, do servidor de imagens HORUS Image Server v2.6.20220417. Esta tarefa recebe e o comando e organiza o acesso as tabelas de dados para processamento do comando.

Figura A.5: HORUSSERVER – COMMAND EXECUTER

A.2.4 HORUSSERVER – TABLE METADATA

A Figura A.6, apresenta os recursos da tabela de metadados, *Table Metadata*, do servidor. Como os dados armazenados no servidor de imagens são distribuídos, esta tabela atua como um descritor da localização das tabelas entre as unidades remotas.

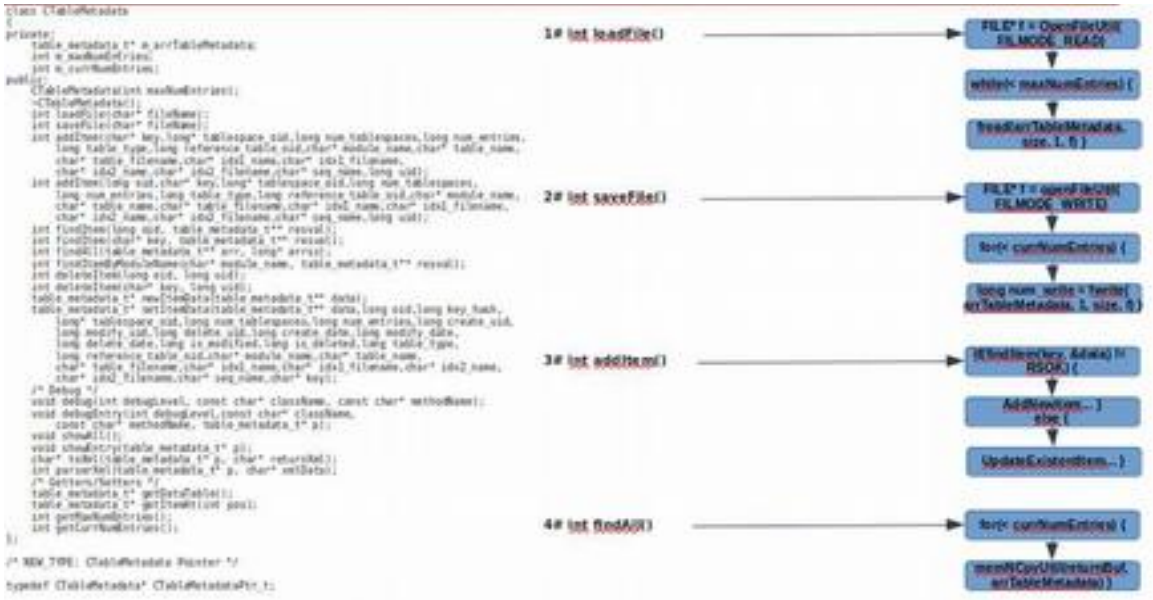

Figura A.6: HORUSSERVER – TABLE METADATA

A.2.5 HORUSSERVER – TABLESPACE

A Figura A.7, apresenta as definições dos espaços das tabelas, *Tablespace*. O espaço de tabelas são áreas de armazenamento distribuídas pelas unidades remotas. A estrutura de metadados do servidor informa o espaço de tabela no qual os dados se encontram. Este espaço de tabela está associado a um grupo de unidades de armazenamento e os dados são armazenados de forma distribuída no nível de blocos pelos equipamentos que fazem parte de um mesmo grupo de unidades remotas.

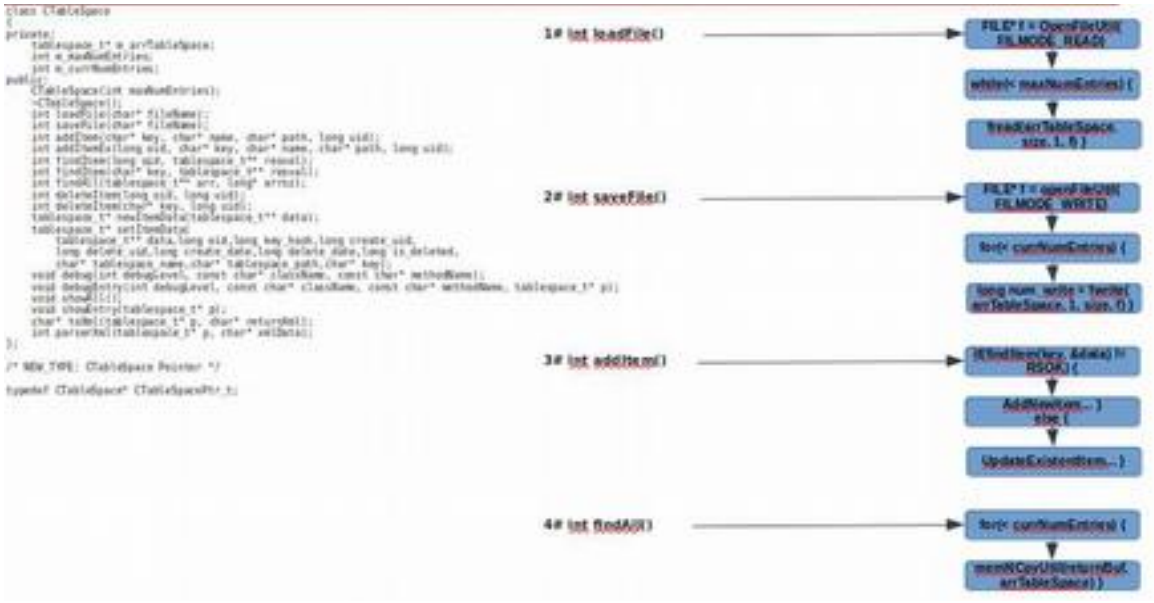

Figura A.7: HORUSSERVER – TABLESPACE

A.2.6 HORUSSERVER – REMOTE DISPATCHER

A Figura A.8, apresenta o despacho de tarefas remotas, *Remote Dispatcher*, que envia para um grupo de unidades remotas, a solicitação de seleção, processamento e retorno dos dados.

Figura A.8: HORUSSERVER – REMOTE DISPATCHER

A.2.7 HORUSSERVER – RETURN RESULTS

A Figura A.9, apresenta a estrutura de dados básica para retorno dos dados.

Figura A.9: HORUSSERVER – RETURN RESULTS

A.3 HORUSWORKER

O servidor de imagens HORUS Image Server v2.6.20220417, possui dois módulos distintos. O modulo servidor, HORUSSERVER, que executa no servidor hospedeiro e o modulo de trabalho, HORUSWORKER, que executa nas unidades de armazenamento CSD, e será discutido nas próximas seções.

A.3.1 HORUSWORKER – WORKER LISTENER

A Figura A.10, apresenta o fluxo de dados da tarefa de monitoramento do canal de comunicação, *Worker Listener*, do servidor de imagens HORUS Image Server v2.6.20220417. Esta tarefa fica aguardando a chegada de solicitações do módulo servidor, e quando recebe uma solicitação, dispara uma tarefa para execução.

Figura A.10: HORUSWORKWR – WORKER LISTENER

A.3.2 HORUSWORKER – DATA TABLE

A Figura A.11, apresenta os recursos da tabela de dados, *Data Table*, do servidor.

Figura A.11: HORUSWORKWR – DATA TABLE

A.3.3 HORUSWORKER – IMAGE TABLE

A Figura A.12, apresenta os recursos da tabela de armazenamento das referências das imagens, *Image Table*, do servidor.

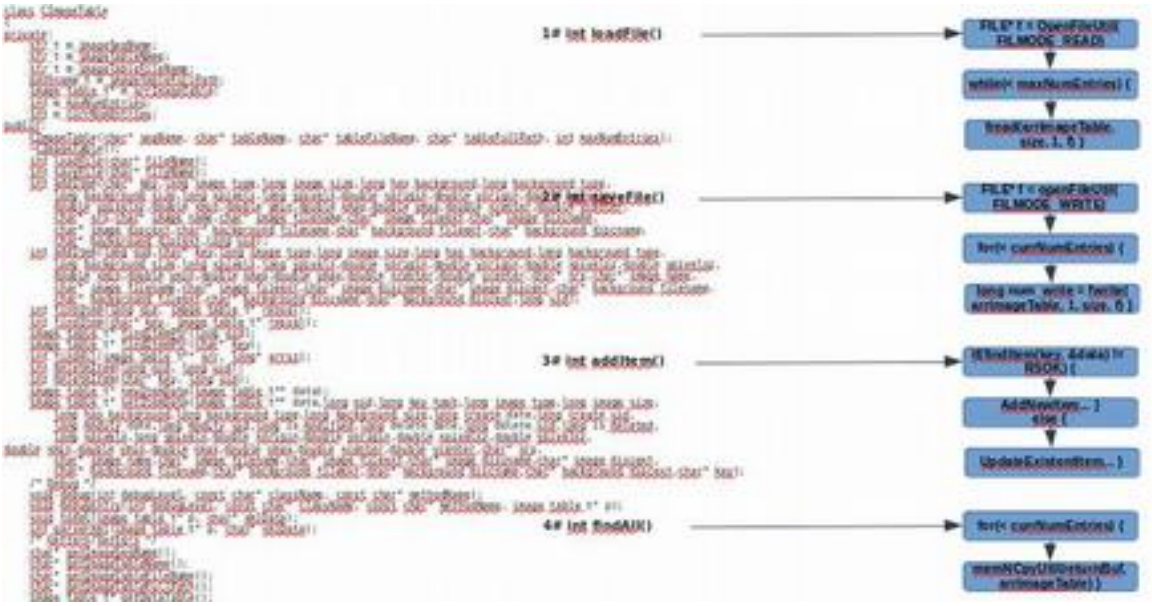

Figura A.12: HORUSWORKWR – IMAGE TABLE

A.3.4 HORUSWORKER – FILE LOCATOR (DIRECTORY)

A Figura A.13, apresenta a lista de diretórios que localiza os arquivos das imagens nas unidades de armazenamento CSD. Esta lista de diretórios fica armazenada na unidade de armazenamento CSD responsável por controlar a distribuição de dados no nível de bloco e efetuar a junção dos dados para processamento.

Figura A.13: HORUSWORKWR – FILE LOCATOR (DIRECTORY)

A.3.5 HORUSWORKER – BLOCK LOCATOR

A Figura A.14, apresenta a estrutura do localizador dos blocos dos arquivos de dados que estão espalhados por um grupo de unidades de armazenamento CSD. Esta lista de blocos fica armazenada na unidade de armazenamento CSD responsável por controlar a distribuição de dados no nível de bloco e efetuar a junção dos dados para processamento.

Figura A.14: HORUSWORKWR – BLOCK LOCATOR

A.3.6 HORUSWORKER – REMOTE FILE REQUEST

A Figura A.15, apresenta a estrutura responsável por efetuar as solicitações dos blocos de dados para as unidades remotas que participam de um mesmo grupo.

Figura A.15: HORUSWORKWR – REMOTE FILE REQUEST

A.3.7 HORUSWORKER – DATA PROCESSOR

A Figura A.16, apresenta a estrutura responsável pelo processamento dos dados. Após efetuar as solicitações dos blocos de dados de um arquivo para as unidades remotas que participam de um mesmo grupo. Os blocos de dados são organizados em memória e compartilhados usando *shared memory* com os módulos de processamento de conversão, projeção e simplificação da imagem.

igura A.16: HORUSWORKWR – DATA PROCESSOR

A.3.8 HORUSWORKER – RETURN DATA RESULT

A Figura A.17, apresenta a estrutura de dados básica para retorno dos dados.

Figura A.17: HORUSWORKWR – RETURN DATA RESULT

A.3.9 HORUSWORKER – FILE WRITER

A Figura A.18, apresenta a escrita dos dados distribuída e *multi thread*.

Figura A.18: HORUSWORKWR – FILE WRITER

Bibliografia

[1] A. De Mauro, M. Greco, M. Grimaldi; "A formal definition of Big Data based on its essential features"; Library Review Vol. 65 No. 3, 2016; pp. 122-135

[2] A. De Mauro, M. Greco, M. Grimaldi; "Understanding Big Data Through a Systematic Literature Review: The ITMI Model"; International Journal of Information Technology & Decision Making Vol. 18, No. 4 (2019); p1433–1461

[3] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E. M. Nguifo; "An experimental survey on big data frameworks"; Future Generation Computer Systems 86 (2018); p546–564

[4] P. Pääkkönen, D. Pakkala; "Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems"; Big Data Research 2 (2015); p166–186

[5] Apache Spark – Lightning -fast unified analytics engine, https://spark.apache.org/

[6] Apache Spark - API MLlib – Scalable machine learning library, https://spark.apache.org/mllib/

[7] Apache Spark - API GraphX – API for graphs and graph-parallel computation, https://spark.apache.org/graphx/

[8] L. Chun-Yi, B. K. Jagadish, J. Myoungsoo, T. K. Mahmut; “PEN: Design and Evaluation of Partial-Erase for 3D NAND-BasedHigh Density SSDs”; ISCA 2020.

[9] W. He and D. H. C. Du; “SMaRT: An Approach to Shingled Magnetic Recording Translation”; 15th USENIX Conference on File and Storage Technologies, 2015.

[10] Y. Li, H. Wang, X. Zhang, N. Zheng, S. Dahandeh and T. Zhang; “Facilitating Magnetic Recording Technology Scaling for Data Center Hard Disk Drives through Filesystem-level Transparent Local Erasure Coding”; 15th USENIX Conference on File and Storage Technologies, 2015.

[11] M. Bjørling, J. González and P. Bonnet; “LightNVM: The Linux Open-Channel SSD Subsystem”; 15th USENIX Conference on File and Storage Technologies, 2015.

[12] Anonymous; “I/O and Energy Efficient Large-scale Image Similarity Search Using Computational Storage Devices”, USENIX FAST.

[13] D. Jaeyong, C. F. Victor, B. Hossein, T. Mahdi, R. Siavash, H. Ali, Heydarigorji, S. Diego, F. G. Bruno, S. Leandro, K. S. Min, M. V. L. Priscila, M. G. F. França, A. Vladimir; “Cost-effective, Energy-efficient, and Scalable Storage Computing for Large-scale AI Applications”; ACM Transactions on Storage, Vol. 16, No. 4, Article 21. Publication date: October 2020.

[14] C. Wei, L. Yang, C. Zhushi, Z. Ning, L. Wei, W. Wenjie, O. Linqiang,W. Peng, W. Yijing, K. Ray, L. Zhenjun, Z. Feng, Z. Tong; “POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database”; 18th USENIX Conference on File and Storage Technologies; pages 29-41; 2020.

[15] H. Ali, T. Mahdi, R. Siavash, B. Hossein; “STANNIS: Low-Power Acceleration of Deep Neural Network Training Using Computational Storage Devices”; 2020.

[16] Israel de Andrade; “Como Funcionam os Carros Autonomos”; https://medium.com/brasil-ai/como-funcionam-os-carros-autônomos-parte-1-sensoriamento-e-visão-computacional.

[17] E. Passos; “Tesla agora sabe ultrapassar sozinho e dirigir na completa escuridão”; Revista Quatro Rodas; https://quatrorodas.abril.com.br/noticias/tesla-agora-sabe-ultrapassar-sozinho-e-dirigir-na-completa-escuridao/.

[18] Rosa, Marcelo Carvalho; Sistema de Direção Elétrica Assistida para Veículos Elétricos e Híbridos Utilizando Motor de Indução [Curitiba] 2015 (UTFPR, Especialização em Sistemas Embarcados para Industria Automotiva, 2015); Trabalho de Conclusão de Curso; Universidade Tecnológica Federal do Paraná – UTFPR; 1. Direção elétrica assistida; 2. Veículos híbridos; 3. Eficiência na emissão de gases.

[19] GDAL: Translator library for raster and vector geospatial data formats, https://gdal.org/.

[20] Oracle Spatial: User's Guide and Reference, Release 9.2, March 2002.

[21] PostGIS 3.1.8: Development Manual, August 2022.

[22] Y. Fang, M. Friedman, G. Nair, M. Rys, A. E. Schmidt, "Spatial Indexing in Microsoft SQL Server 2008", Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008

[23] Y. LeCun, “Learning Process in an Asymmetric Threshold Network”, 1986.

[24] Y. LeCun, “A Theoretical Framework for Back-Propagation”, 1988.

[25] D. Eigen, J. Rolfe, R. Fergus, Y. LeCun, “Understanding Deep Architectures using a Recursive Convolutional Network”, 2014.

[26] T. Surat, M. Bradley, H. T. Kung; “Distributed Deep Neural Networks over the Cloud, the Edge and End Devices”; Naval Postgraduate School Agreements No. N00244-15-0050 and No. N00244-16-1-0018. 2015.

[27] Z. Li, W. Hao, T. Radu, and H. C. David Du; “Distributing Deep Neural Networks with Containerized Partitions at the Edge”; 2019.

[28] C. Sandeep, C. Eyal, P. Evgenya, C. Thianshu, K. Sachin; “Neural Networks Meet Physical Networks: Distributed Inference Between Edge Devices and the Cloud”; 2018.

[29] B. Eunjin, K. Dongup, and K. Jangwoo; “A Multi-Neural Network Acceleration Architecture”; ISCA 2020.

[30] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011, https://scikit-learn.org/stable/user_guide.html.

[31] Keras: The Python Deep Learning library, F. Chollet et al, 2015, https://keras.io/.

[32] TensorFlow, J. Dean, G. Corrado, A. Ng., 2011, https://www.tensorflow.org/.

[33] C. Chunlei, C. Li, Z. Lei, Z. Xiaoyun, G. Zhiyong; “RCDFNN: Robust Change Detection Based on Convolutional Fusion Neural Network”; 2018.

[34] Y. S. Abu-Mostafa, M. Magdon-Ismail, H. Lin Authors, “e-Chapter 8 – Support Vector Machines” in Learning From Data a Short Course, Country: Passadena, CA, USA, 2012, ch. 8.

[35] L. A. Lim, H. Y. Keles, “Learning Multi-scale Features for Foreground Segmentation”, 2018

[36] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, H. Vincent; "Federated learning for internet of things: A comprehensive survey."; IEEE Communications Surveys & Tutorials 23.3; 2021; p1622-1658.

[37] V. Hedge, S. Usmani; "Parallel and Distributed Deep Learning"; May 31, 2016; p1-8

[38] L. Dutta, S. Bharali; "Tinyml meets iot: A comprehensive survey. Internet of Things", v. 16, p.100461, 2021.

[39] T. Meng, X. Jing, Z. Yan, W. Pedrycz; "A survey on machine learning for data fusion."; Information Fusion 57 (2020); p115-129.

[40] X. Ding, X. Jing, Z. Yan, L. T. Yang; "A survey on data fusion in internet of things: Towards secure and privacy-preserving fusion."; Information Fusion 51 (2019); p129-144.

[41] B. P. L. Lau, S. H. Marakkalage, Y. Zhou, N. U. Hassan, C. Yuen, M. Zhang, U. X. Tan; "A survey of data fusion in smart city applications."; Information Fusion 52 (2019); p357-374

[42] D. G. Massimo, G. Maurizio; “WiSARD rp for Change Detection in Video Sequences”; ESANN 2017.

[43] Y. S. Abu-Mostafa, M. Magdon-Ismail, H. Lin Authors, “e-Chapter 7 – Neural Networks” in Learning From Data a Short Course, Country: Passadena, CA, USA, 2012, ch. 7.

[44] L. A. de Araujo; "RWiSARD: Um Modelo de Rede Neural Sem Peso para Reconhecimento e Classificaçao de Imagens em Escala de Cinza; Rio de Janeiro; UFRJ/COPPE, 2011.

[45] D. G. Massimo, G. Maurízio; “Change Detection with Weightless Neural Networks”; CVPR 2014.

[46] CDnet: ChangeDetection.Net, 2012-2014, http://changedetection.net/

[47] C. G. B. Caio; “An Adaptation of the Data Flow Library Sucuri Static Scheduler for In-Situ Computing”; Dissertation presented to COPPE/UFRJ in March, 2018.

[48] R. J. N. Silva, "Estratégias de Escalonamento para Ambientes de Execução Guiada por Fluxo de Dados", Tese de Mestrado, 2016, UERJ.

[49] Mitchell, Tyler. “Web Mapping Illustrated.” (2005).

[50] K. Jeremy, A. Willian, B. Willian, B. Nadya, B. Robert, B. Chansup, C. Gary, G. Kenneth, H. Matthew, K. Jonathan, M. Andrew, M. Peter, P. Andrew, R. Albert, R. Antonio, Y. Charles; “Dynamic Distributed Dimensional Data Model (D4M) Database and Computation System”; 2012.

[51] T. M. Özsu, P. Valduriez; "Principles of distributed database systems."; 2011; p845.

[52] T. Arash, G. L. Juan, S. Mohammad, G. Saugata, M. Onur; “MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices”; ISCA 2020.

[53] J. Huang, S. Nath, A. Badam, S. Sengupta, B. Sharma, L. Caulfield and M. K. Qureshi; “FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs”; 15th USENIX Conference on File and Storage Technologies, 2015.

[54] R. G. Tinedo, J. Sampé, E. Zamora, M. S. Artigas and P. G. López; “Crystal: Software-Defined Storage for Multi-tenant Object Stores”; 15th USENIX Conference on File and Storage Technologies, 2015.

[55] S. K. Lee, K. H. Lim, H. Song, B. Nam, S. H. Noh; “WORT: Write Optimal Radix Tree for Persistent Memory Storage Systems”; 15th USENIX Conference on File and Storage Technologies, 2015.

[56] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien and H. S. Gunawi; “Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs”; 15th USENIX Conference on File and Storage Technologies, 2015.

[57] F. Douglis, A. Duggal, P. Shilane, T. Wong, S. Yan and F. Botelho; “The Logic of Physical Garbage Collection in Deduplicating Storage”; 15th USENIX Conference on File and Storage Technologies, 2015.

[58] PCI Express Group; "PCI Express Base Specification"; Revision 2.0; December 20, 2006

[59] NVM Express Group; "NVM Express Base Specification"; Revision 2.0; May 13th, 2021

[60] Toshiba; "Flash Memory"; Semiconductor Catalog; Mar. 2016

[61] R. Zelenovsky, A.Mendonça; "Hardware e Interfaceamento"; Editora: MZ Editora; Ano: 2002

[62] ARM Processors, "ARM Cortex-A57 MPCore Processor Technical Reference Manual"; 2016.

[63] ARM Processors, "ARM Cortex-A53 MPCore Processor - Technical Reference Manual"; Revision: r0p3; 2014

[64] ARM Processors, "ARM Cortex-M7 Processor - Technical Reference Manual"; Revision r0p2; 2014

[65] ARM Processors, "ARM Cortex-A7 MPCore - Technical Reference Manual"; Revision: r0p5; 2013

[66] L. Torvalds; "Linux: a Portable Operating System"; UNIVERSITY OF HELSINKI - Department of Computer Science; Helsinki January 31, 1997

[67] Citrix; "Citrix XenServer 7.1 LTSR - Citrix Product Documentation"; August 14, 2022

[68] Oracle; Oracle VM - Concepts Guide for Release 3.3"; 2022; https://docs.oracle.com/cd/E50245_01/E50249/html/vmcon-preface.html

[69] Oracle; Oracle VM - Manager User's Guide for Release 3.3"; 2022; https://docs.oracle.com/cd/E50245_01/E50250/html/index.html

[70] NGD Systems; "Newport Platform Product Brief"; 2020

[71] Xilinx - "SmartSSD Computational Storage Drive - Product Brief"; 2022

[72] A. Bulent, B. Bart, R. John, K. Mathias, M. Ashutosh, B. A. Craig, S. Bedri, B. Alper, J. Christian, J. S. Willian, M. Haren, and W. Charlie; “Data Compression Accelerator on IBM POWER9 and z15 Processors”; ISCA 2020.

[73] HENNESSY, J., PATTERSON, D., “Computer Architecture: A Quantitative Approach”, Morgan-Kaufmman, pp 29-38 e pp 76-83, 1997.

[74] J. J. B. Lima; “Sistema Antibloqueio (ABS) para Freios Eletromecânicos utilizando Controle por Modos Deslizantes”; Tese de Mestrado – Universidade Federal do Rio de Janeiro; 2005.

[75] Trimble Navigation Limited; Mobile Positioning and Communications; CrossCheck GSM 1900 with IQ Event Engine; Part Number: 43458-00 – Revisão: C – Data: Fevereiro, 2001, https://www.trimble.com.

[76] Trimble Navigation Limited; Mobile Positioning and Communications; CrossCheck GSM/GPRS 850/1900 Installation Manual; Part Number: 47770-10-ENG – Revisão: A - Data: Janeiro, 2004, https://www.trimble.com.

[77] A. J. Fornasari, D. Biondo, L. S. Roani; “Sistema de Controle de Temperatura de Ar Condicionado Automotivo”; UTFPR, Tecnologia em Automação Industrial, 2013.

[78] C. E. Milhor, L. C. Passarini; “Estado da Arte do Controle Eletrônico dos Motores de Combustão Interna”; II Congresso Nacional de Engenharia Mecânica; João Pessoa – PB; Agosto, 2002.

[79] C. M. Rodrigo, O. T. Giovane, L. P. Maurício, L. P. Laércio, T. C. Amarildo, M. G. F. Felipe; “Value Reuse Potential in ARM Architectures”; IEEE 28th International Symposium on Computer Architecture and High Performance Computing; 2016.

[80] L. P. Maurício, R. C. Bruce, T. C. Amarildo, M. G. F. Felipe, O. A. N. Philippe, “A Speculative Trace Reuse Architecture with Reduced Hardware Requirements”, 2006.

[81] L. M. F. A. VIANA, Memorização Dinâmica de Traces com Reuso de Valores de Instruções de Acesso à Memória [Rio de Janeiro] 2002, XIT, 118 p. 29,7 cm (COPPE/UFRJ, MSc., Engenharia de Sistemas e Computação, 2002), Tese - Universidade Federal do Rio de Janeiro, COPPE, 1. Reuso Dinâmico de Traces, 2. Arquitetura de Processador, I. C0PPEíWR.J 11. Título (série).