CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA CELSO SUCKOW DA FONSECA

DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

ENGENHARIA ELETRÔNICA

PEDRO AZEVEDO DA CONCEIÇÃO

ESTUDO SOBRE FPGAS EM DATACENTERS DE ALTO VOLUME DE DADOS: UMA PROPOSTA PARA CONTROLE DE DADOS DE EMPRESAS NO RIO DE JANEIRO

TRABALHO DE CONCLUSÃO DE CURSO

RIO DE JANEIRO

2025

PEDRO AZEVEDO DA CONCEIÇÃO

ESTUDO SOBRE FPGAS EM DATACENTERS DE ALTO VOLUME DE DADOS: UMA PROPOSTA PARA CONTROLE DE DADOS DE EMPRESAS NO RIO DE JANEIRO

Trabalho de Conclusão de Curso de Engenharia Eletrônica (GELT), do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca.

Orientador: Prof. Dr. João Roberto de Toledo Quadros

RIO DE JANEIRO

2025

Espaço destinado a elaboração da ficha catalográfica sob responsabilidade exclusiva da Biblioteca Central do CEFET/RJ.

AGRADECIMENTOS

Primeiramente agradeço a Deus, que me fortaleceu e permitiu com que eu pudesse ingressar, estudar e concluir a graduação.

Agradeço meus pais, pelo apoio que a todo momento me deram, desde minha infância até os dias de hoje.

Agradeço meus professores João Roberto de Toledo Quadros e Luciana Faletti Almeida, por proporcionarem aulas didáticas, compatíveis com a demanda do mercado e por oferecerem todo o suporte necessários para o meu desenvolvimento, assim como para demais alunos.

Também gostaria de agradecer aos meus colegas da turma de eletrônica 2018.2 por estarmos unidos nos estudos, durante toda minha trajetória pela graduação.

RESUMO

CONCEIÇÃO, Pedro Azevedo. **ESTUDO SOBRE FPGAS EM DATACENTERS DE ALTO VOLUME DE DADOS: UMA PROPOSTA PARA CONTROLE DE DADOS DE EMPRESAS NO RIO DE JANEIRO.** 2025. 40 páginas. Trabalho de Conclusão de Curso – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca. Rio de Janeiro. Rio de Janeiro, 2025.

Este trabalho apresenta um estudo sobre a implementação de *FPGAs* em data centers para o processamento de grandes volumes de dados em estudos de *Deep Learning*. Visando maior eficiência energética, menor latência e menor consumo de recursos de chip, foi elaborada uma proposta de *FPGAs* otimizados para uso de *LSTMs*. Além disso, também foi abordado um método de comunicação *RFID* com criptografia para garantir segurança na comunicação entre os dispositivos. A fim de exemplificar, foi proposta uma aplicação do hardware no cálculo de previsão de demanda elétrica das empresas sediadas no Município do Rio de Janeiro.

Palavras-chave: *Deep Learning*, *LSTM*, *FPGA*, Previsão de demanda por energia, *RFID*, Criptografia, *forecasting*.

ABSTRACT

CONCEIÇÃO, Pedro Azevedo. **ESTUDO SOBRE FPGAS EM DATACENTERS DE ALTO VOLUME DE DADOS: UMA PROPOSTA PARA CONTROLE DE DADOS DE EMPRESAS NO RIO DE JANEIRO.** 2025. 40 páginas. Trabalho de Conclusão de Curso – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca. Rio de Janeiro. Rio de Janeiro, 2025.

This work presents a study on the implementation of FPGAs in data centers for processing large volumes of data in Deep Learning studies. Aiming for greater energy efficiency, lower latency and less consumption of chip resources, a proposal was made for FPGAs optimized for use in LSTMs. In addition, a method of RFID communication with cryptography to guarantee secure communication between devices was also discussed. As an example, a hardware application was proposed for calculating electricity demand forecasts for companies based in the city of Rio de Janeiro.

Keywords: Deep Learning, LSTM, FPGA, forecasting electricity demand, RFID, Criptografia.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

BRAM	*Block Random Access Memory*
CCEE	Câmara de Comercialização de Energia Elétrica
CLK	Clock
CPU	Computer Processing Unit
DSP	Digital Signal Processors
FIFO	First In First Out
FPGA	Field-Programable Gate Array
GPU	Graphics Processing Unit
HLS	High-Level Synthesis
IA	Inteligência Artificial
LFSR	Linear Feedback Shift Register
LSTM	Long-Short Term Memory
LUT	Look-Up Table
LUTRAM	Look-Up Tale RAM
MAC	Multiplier-accumlator
NLP	Natural Language Processing
PE	Processing Element
RAM	Random Access Memory
RFID	Radio Frequency Identification Device
RNN	Recurrent Neural Network
TEA	Tiny Encryption Algorithm
XTEA	*Extended Tiny Encryption Algorithm*

LISTA DE SÍMBOLOS

b_f	Bias na porta Forget f
b_g	Bias na porta da célula g
b_i	Bias na porta de entrada i
b_o	Bias na porta de saída o
c	Estado da célula c
c_t	Estado da célula no passo t
D_T	Texto descriptografado
E_T	Texto criptografado
f	Porta Forget
f_t	Porta Forget no passo t
g	Porta da célula
g_t	Porta da célula no passo t
h	Saída da célula
h_t	Saída da célula no passo t
i	Porta de entrada
i_t	Porta de entrada no passo t
K	Valor da chave
K_0	Saída da etapa de agendamento de chaves
m	Modo do algoritmo *XTEA*
o	Porta de saída
o_t	Porta de saída no passo t
p_o	Polinômio 0 da sequência polinomial
P_T	Chave de 128bit do *XTEA*
P_1	Variável que recebe 32bit de P_T
P_2	Variável que recebe 32bit de P_T
RF	Função rodada
RF_0	Função rodada de saída
RF_i	Função rodada de entrada
t	Passo atual do processo

W_*	Matriz peso
W_{hf}	Matriz peso da sequência de input h_f na porta Forget f
W_{hg}	Matriz peso da sequência de input h_g na porta da célula g
W_{hi}	Matriz peso da sequência de input h_i na porta de entrada i
W_{ho}	Matriz peso da sequência de input h_o na porta de saída o
W_{xf}	Matriz peso da sequência de input x_f na porta Forget f
W_{xg}	Matriz peso da sequência de input x_g na porta da célula g
W_{xi}	Matriz peso da sequência de input x_t na porta de entrada i
W_{xo}	Matriz peso da sequência de input x_o na porta de saída o
X_D	Equação usada no processo de descriptografia
X_E	Equação usada no processo de criptografia
x_t	Sequência de sinais de entrada no passo t
α_1	Constante determinada para uso no cálculo de X_E
δ_1	Constante determinada para uso no cálculo de X_E
δ_2	Constante determinada para uso no cálculo de X_D
+	Soma de elementos
⊙	Multiplicação de elementos
^	Operador lógico *XOR*
&	Operador lógico *AND*
≫	Deslocamento à direita de bits
≪	Deslocamento à esquerda de bits

SUMÁRIO

1 INTRODUÇÃO

Desde que começou a popularização do uso da Internet, em meados dos anos 90 do século passado, uma gama de aplicações surgiu a partir de seu uso e das adaptações sobre seus mecanismos (Raghavan e Perera, 2017).

Uma das tecnologias que mais avançaram nesse novo quadro foi o de armazenamento de informação. A princípio, o acesso a servidores que pudessem gravar e armazenar uma grande quantidade de dados se limitava a dados textuais (Raghavan e Perera, 2017). Ao longo do tempo, com a melhoria na tecnologia de transmissão de dados, inclusive com a introdução da fibra ótica, os dados se tornaram mais complexos e menos textuais.

O ambiente da internet viu-se diante do dilema de guardar imagens, vídeos, sons e todo tipo de informação baseado em mídias diferentes, sendo necessário criar novas estruturas de armazenamento e busca para esses tipos de dados, denominados de não-convencionais (Raghavan e Perera, 2017).

Devido ao exponencial crescimento do fluxo global de dados e informações na internet, adaptou-se conceitos teóricos para o mundo prático, surgindo, em princípio, os *datawarehouse* e atualmente os sistemas de (Chung, Liu e Lee, 2015).

Esses novos sistemas, que permitem armazenamento de grandes quantidades de dados, em vários formatos e mídias, trouxeram novos desafios relacionados à gestão, controle, busca e análise desses grandes volumes de informações (Chung, Liu, e Lee, 2015) (Raghavan e Perera, 2017).

Esses desafios incluem tanto desenvolvimento de novos softwares para tratar de armazenamentos, quanto dados, compostos de estruturas inteligentes, com gerência de dados baseados em conceitos de *Machine Learning* e *Deep Learning*, os desafios também incluem necessidade de hardwares mais rápidos e capazes de tratar de grandes volumes, permitindo tomadas de decisão mais eficazes e imediatas (Rouhani, at al, 2015),

Nesse novo contextos, de plataformas de hardware e software voltados para Big Data com *Deep Learning*, uma das pesquisas que visa resolver os problemas que surgem, trata da eficiência dos novos sistemas computacionais e sua visão sustentável de consumo energético (Chung, Liu, e Lee, 2015) (Rouhani, at al, 2015).

As novas estruturas de *data centers*, que são ambientes voltados para essas plataformas, têm se tornado cada vez maiores e mais exigentes, tanto em termo de espaço, quanto de consumo energético, evidenciando que, para aumentar sua escalabilidade, é essencial superar esses desafios, que tendem a ser impeditivos (Hoozemans et al., 2021).

Busca-se, assim, construção de plataformas de baixo consumo, eficientes e capazes de gerenciar esse grande volume de dados, proporcionando uma real eficiência computacional, de hardware e energética.

Para tanto, esse trabalho tem como objetivo primeiro estudar o sobre o hardware do tipo *Field-Programmable Gate Array* (FPGA), focando em sua arquitetura flexível, de baixa latência, baixo consumo energético e alta compatibilidade, observando algumas implementações, algoritmos usados, perfazendo uma abordagem teórica sobre o uso desse mecanismo em ambiente de *Data Center*.

De modo a ter uma pequena abordagem de conteúdo prático, é apresentando também o FPGA como uma hipótese de solução de hardware para *Big Data* e *Machine Learning* utilizando um ambiente de CPU baseado em FPGA, focando em maior velocidade de acesso, eficiência energética e robustez, para análise de consumo energético de empresas no Município do Rio de Janeiro.

A ideia de propor um *Data Center* FPGA para controlar esses dados é justamente por serem dados de grande volume, que prescindem tomadas de decisões rápidas, que, por conta de constante controle (sete dias da semana, 24 horas por dia) tem que ser montados visando baixo consumo e baixa latência nas buscas de informações.

Conforme visto em He et al (2021), o uso de motores de aceleração para dispositivos de controle de *Big Datas*, combinado com um sistema de autenticação de RFID, trabalha com cifras leves, que exigem baixo consumo e fácil tratamento.

Também é visto que os FPGAs permitem a implementação de arquiteturas de hardware altamente otimizadas, adaptadas a aplicativos, eliminando a sobrecarga usada nas arquiteturas tradicionais de computadores. (Hoozemans et al, 2021).

Os FPGAs podem implementar arquiteturas orientadas a fluxo de dados com altos níveis de paralelismo (pipeline) e podem oferecer alta taxa de transferência de

aplicativos, geralmente com alta eficiência energética. (Cong et al., 2022). Os aplicativos sensíveis à latência podem aproveitar os aceleradores de FPGA conectando-se diretamente à camada física de uma rede e realizar transformações de dados sem passar pelas pilhas de software do sistema host (Cong et al., 2022).

Para tanto, a justificativa desse trabalho é contribuir no avanço de estudos práticos sobre uso de plataformas baseadas em FPGA para uso em ambiente de *Big data*. Como exemplo desse uso, já citado, foi organizado uma proposta inicial de uma plataforma dessa natureza para poder gerenciar dados de consumo energético de empresas do Município do Rio de Janeiro, ajudando a tomada de decisões sobre ações de redução de consumo, se possível, com uso integrado a GPUs para localização geográfica.

Esse trabalho está organizado com uma introdução, a fundamentação teórica, estudos sobre o FPGA e seus modelos de aplicação, inclusive com uma visão de segurança com RFID, os detalhes dos dados a serem tratados e uma sugestão hipotética de exemplo de uso do mesmo, além da discussão de resultados previstos, com uma conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata sobre os conceitos fundamentais que fazem parte do embasamento teórico do tema abordado nesse trabalho. Os trabalhos citados foram consultados nas plataformas Google Scholar e Scopus, utilizando as palavras chaves FPGA*, DataMining, Deep learning, Big Data* e *Data Centers*.

2.1 MOTOR DE ACELERAÇÃO EM FPGA PARA *LSTM*

A fim de otimizar o desempenho de hardware em aplicações de Big Data, a utilização de motores de aceleração é essencial tanto do ponto de vista computacional quanto energético. Estudos de He et al. (2021) evidenciam ganhos significativos em capacidade de processamento e redução de latência em algoritmos de *Deep Learning*, como o *Long Short-Term Memory (LSTM)*, quando implementados em *FPGAs*.

2.1.1 *Long Short-Term Memory (LSTM)*

O *Long Short-Term Memory (LSTM)* é um tipo de rede neural projetado para resolver problemas envolvendo sequências longas, pois consegue reter informações relevantes por períodos prolongados. Assim, supera as *recurrent neural networks (RNN)* e redes *feedforward* em tarefas que exigem memória de longo prazo. No entanto, a dependência de feedback do *LSTM* limita o paralelismo em *GPUs* e *CPUs*, tornando os *FPGAs* uma alternativa eficaz para suprir os problemas previamente citados (Cao et al, 2019).

O *LSTM*, no processo de propagação de rede consegue lembrar de atributos importantes e descartar os possivelmente não importantes, por um longo período. A estrutura de uma célula *LSTM* é mostrada na Figura 1. A célula *LSTM* no tempo atual t, recebe uma sequência de entradas x_t do passo atual e a sequência h_{t-1} do passo anterior $t-1$, como entrada e gera a saída h_t como parte da entrada do próximo passo. Este *feedback* continua até a rede chegar no passo especificado T. Cada célula *LSTM* contém memória especial para armazenar o estado atual da rede c_t. Além disso,

contém porta de entrada, porta de célula, porta de saída e uma porta de Forget para apoiar os recursos de armazenamento e descarte (He et al, 2021).

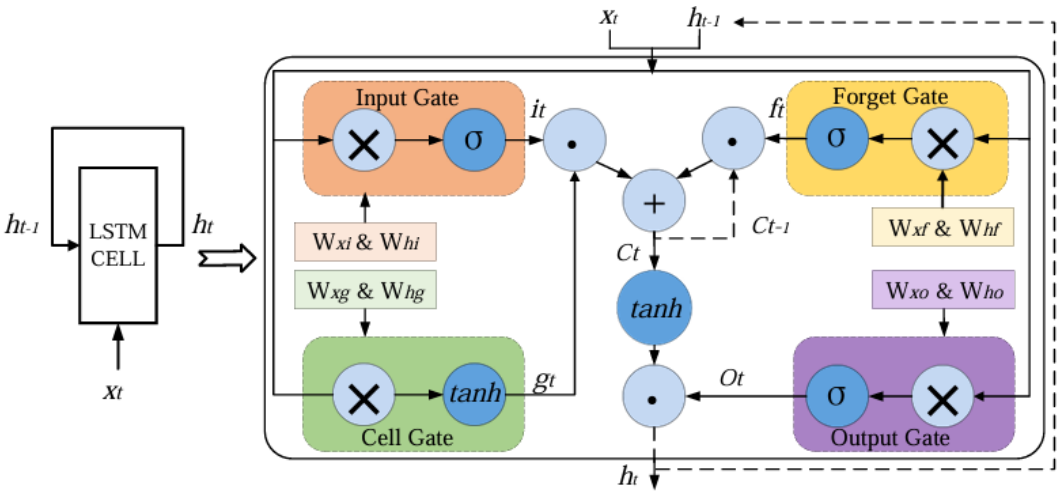

Figura 1 - Célula *LSTM*

Fonte: He et al, 2021

As Fórmulas (1) a (6) descrevem as 4 portas que podem ser utilizadas nesse modelo, sendo essas as seguintes:

$$i_t = \ sigmoid(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i) \quad (1)$$

$$f_t = sigmoid\left(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f\right) \quad (2)$$

$$o_t = sigmoid(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o) \quad (3)$$

$$g_t = tanh(W_{xg}\, x_t + W_{hg}\, h_{t-1} + b_g) \quad (4)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (5)$$

$$h_t = tanh(c_t) \odot o_t \quad (6)$$

Segundo He et al. (2021), i, f, o, g, c e h são respectivamente a porta de entrada (*Input Gate*), porta *forget* (*Forget Gate*), porta de saída (*Output Gate*), porta de célula (*Cell Gate*) célula de estado (*Cell State*) e célula de saída (*Output Cell*) e seus comprimentos são iguais. Sendo t o passo atual e $t-1$ o passo anterior.

O operador $+$ representa adição de elemento e o operador $\odot$ multiplicação de elementos. W_* corresponde ao peso da matriz e b_* o vetor de *bias.* Se o tamanho do vetor de entrada for I, então o tamanho do vetor de *bias* b e o tamanho do vetor de

saída serão ambos N , por fim o tamanho do peso das matrizes W_{x*} e W_{h*} são $N \times I$ e $N \times N$, respectivamente.

2.1.2 Implementação e Otimização do *LSTM*

O acelerador de *LSTM*, também proposto por He et al. (2021), é implementado de forma modular e é estruturado em 3 estágios, conforme Figura 2.

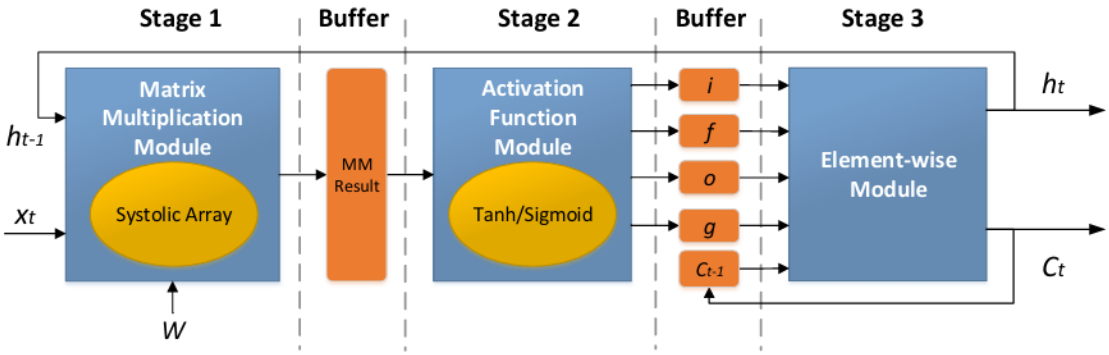

Figura 2 - A estrutura de aceleração *LSTM* proposta.

Fonte: He et al, 2021

2.1.2.1 Módulo de multiplicação matricial

A entrada h_{t-1}, x_t e a matriz de peso W são lidas e a multiplicação matricial é processada usando o algoritmo de matriz sistólica. Após este processo o resultado é armazenado temporariamente na memória global do chip (Cao et al., 2019).

2.1.2.1.1 *Multiplicação matricial*

Segundo He et al. (2021) e de acordo com as Fórmulas (1)-(6), a computação das 4 portas da camada LSTM, inclui 8 pares de multiplicação de matriz-vetor: $W_{xi}x$, $W_{xf}x$, $W_{xo}x$, $W_{xg}x$, $W_{hi}h$, $W_{hf}h$, $W_{ho}h$, $W_{hg}h$. A substituição de transposição nas Fórmulas (1) a (6) permite a matriz multiplicação ser substituída por $(x^T\ h^T)\begin{pmatrix} W_{x*}^T \\ W_{h*}^T \end{pmatrix}$.

Em outras palavras, x e h podem ser combinados em um vetor de entrada de tamanho K e $K = I + N$, e W_{x*} e W_{h*} foram combinados em uma matriz peso de tamanho $K \times N$.

Tomando o *batch size* como M, a operação se torna uma multiplicação matricial, ou seja, processando os vetores de entrada de M um por vez, com a matriz de entrada $M \times K$ e a matriz peso de tamanho $K \times N$. A multiplicação Matricial é o processo que mais consome tempo do sistema inteiro, portanto, o algoritmo de matriz sistólica é implementado justamente para acelerar a computação. (Wang et al, 2019).

2.1.2.1.2 *Particionamento e reordenamento de matriz*

Multiplicação de matrizes em redes neurais frequentemente podem ser aceleradas por processamento em paralelo. Entretanto, em caso de matrizes pesos grandes e recursos computacionais limitados, significa que todas as computações não podem ser finalizadas ao mesmo tempo. Para se utilizar a matriz sistólica para aceleração paralela, a matriz de entrada e a matriz peso, devem ser divididas previamente em pequenos blocos, conforme exibido na Figura 3.

Na Figura 3, presume-se que o tamanho da matriz de entrada e a matriz peso são $M \times N$. A matriz de entrada é dividida em pequenos blocos $m \times k$, a matriz peso é dividida em pequenos blocos $k \times n$ e a matriz saída resulta em pequenos blocos $m \times n$, onde $m = M/TILE\_SIZE$, $k = K/TILE\_SIZE$, $n = N/TILE\_SIZE$ e $TILE\_SIZE$ é o tamanho unidimensional da matriz sistólica. Para garantir que as linhas e colunas da matriz são divisíveis por TILE_SIZE, alguns zeros adicionais são adicionados na matriz entrada como preenchimento, caso necessário.

Nos processos de síntese, do Vivado HLS, a ferramenta de desenvolvimento do Xilinx FPGA (Gafsi et al, 2021), a matriz peso que estiver consumindo bastante recurso, será armazenada na BRAM, como memória global. Quando usada na computação, a matriz peso é lida da memória global, para a memória local da célula (LUTRAM).

Se todos os dados forem lidos em ordem de coluna, haverá descontinuidade, afetando gravemente a eficiência da leitura dos dados. Portanto, é necessário

reordenar os elementos da matriz, como mostrado na parte de baixo da Figura 3; os blocos matriciais na matriz são organizados na mesma ordem que a ordem de leitura da computação (Gafsi et al, 2021).

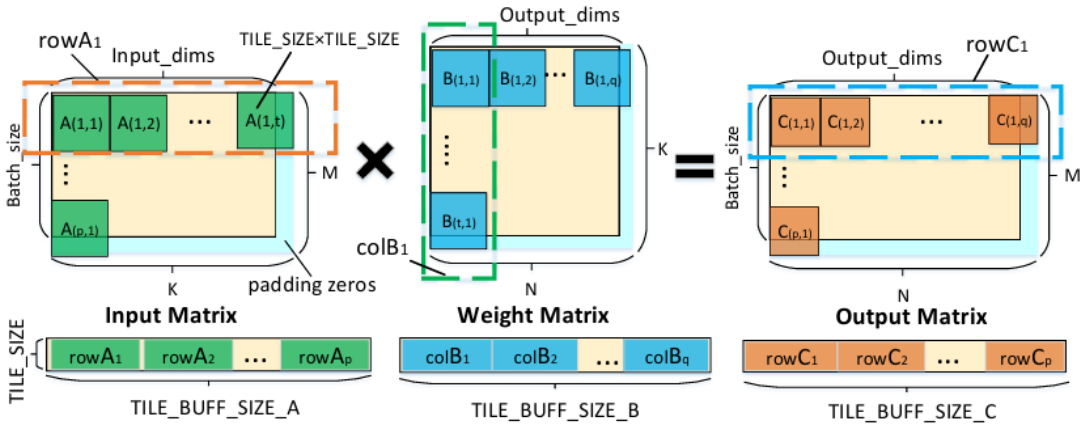

Figura 3 - Processo de particionamento e reordenamento matricial

Fonte: Gafsi et al. 2021, He et al 2021

2.1.2.1.3 *Matriz sistólica*

Segundo H. T Kung (1982), matriz sistólica é descrita como uma matriz para muitos cálculos densos de álgebra linear em uma arquitetura de computação paralela. O principal ponto sobre o uso da matriz sistólica é aumentar a frequência de operação, por meio da redução de número de acessos à memória no fluxo de dados.

Na Figura 4 está a arquitetura de uma matriz sistólica. Cada *Processing Element (PE)*, realiza seus cálculos sem precisar consultar a memória. Apenas o primeiro *PE* acessa a memória diretamente. Após os cálculos de processamento, o resultado é passado adiante para o próximo *PE*. Após cada passo de *clock*, o primeiro *PE* consulta novos dados na memória, até que o último *PE* registre seu resultado na memória (He et al, 2021).

Por rodarem em paralelo, o custo de leitura de memória é reduzido pelas múltiplas operações das *PEs* na matriz. Por fim, resultando em maior fluxo de dados, com menor custo de banda e reduzindo latência que normalmente ocorre em modelos computacionais tradicionais (Gafsi et al, 2021).

Um modelo de arquitetura da matriz sistólica de uma aplicação visto em He et al. (2021) é mostrada na Figura 4. Seu funcionamento detalhado é descrito pelo Algoritmo 1 contido na Figura 5. Cada *PE* contém 3 registradores de armazenamento e uma unidade *multiplier-accumulator (MAC)*, em cada porta de entrada, peso e saída os registradores são utilizados para armazenar os dados de entrada do *PE* da esquerda.

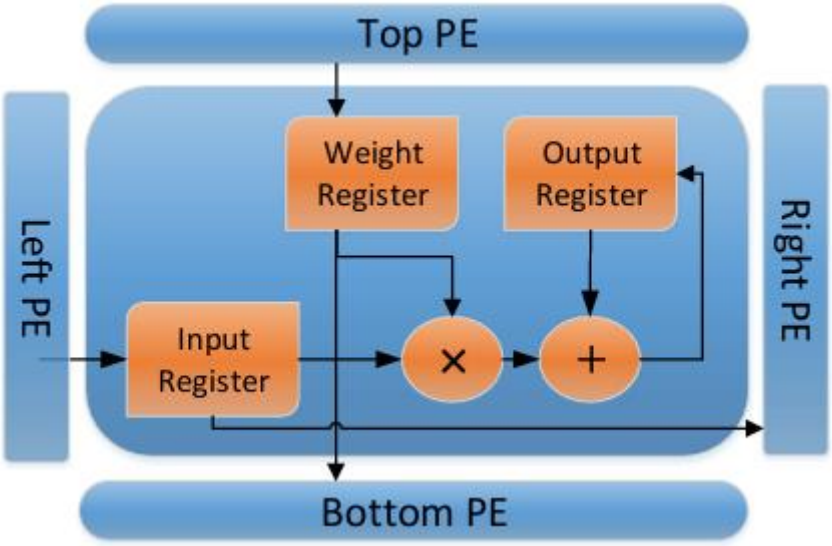

Figura 4 - Estrutura de um *PE* em uma Matriz Sistólica

Fonte: He et al, 2021

O peso do dado do *PE* de cima, e a saída temporária, respectivamente, e o *MAC* executa diversas operações de multiplicação e adição relacionadas aos 3 valores anteriormente citados, a cada ciclo de *clock*. A implementação da matriz sistólica é apresentada no Algoritmo 1 na Figura 5, onde TILE_SIZE, TILE_BUFF_SIZE_A, TILE_BUFF_SIZE_B, TILE_BUFF_SIZE_C da Figura 2, foram abreviados como s, $size\_A$, $size\_B$ e $size\_C$, respectivamente (He, et al, 2021).

Algorithm 1 Systolic Array Algorithm

Input: input matrix: $A = \{a_{ij} | 0 \leq i < s, 0 \leq j < size\_A\}$; weight matrix: $B = \{b_{ij} | 0 \leq i < s, 0 \leq j < size\_B\}$; Number of input matrix column: a_col; No. matrix row being calculated: $\_rowid$; No. matrix column being calculated: $\_colid$

Output: output matrix: $C = \{c_{ij} | 0 \leq i < s, 0 \leq j < size\_C\}$

```
for k = 0 to a_col-1 do
    #pragma pipline
    for i = 0 to s-1 do
        for j = 0 to s-1 do
            last ← (k == 0)?0 : C[c_rowid][c_colid]
            a_rowid ← i
            a_colid ← (i + c_rowid)/s * a_col + k
            b_rowid ← k * b_col + j + c_colid
            b_colid ← j
            C[c_rowid][c_colid] ← last + A[a_rowid][a_colid] * B[b_rowid][b_colid]
        end for
    end for
end for
```

Figura 5 - Algoritmo da Matriz Sistólica

Fonte: He et al, 2021

A Figura 6 mostra a estrutura da matriz sistólica com $TILE_{SIZE} = 16$. As linhas de entrada e as colunas de peso leem da memória global são organizadas na forma *First In, First Out (FIFO)*, e entra sequencialmente na matriz sistólica, um por um.

Em cada ciclo, o *PE* da borda lê os dados do *FIFO*, enquanto o *PE* interior lê a entrada do *PE* adjacente na esquerda e o dado do peso do *PE* adjacente de cima, finaliza a computação na unidade *MAC* e armazena o resultado intermediário na memória.

Após 16 ciclos, a matriz finaliza todas as computações e escreve o dado resultante de volta a memória global. O fluxo da matriz depende no número de *Digital Signal Processors (DSPs)* disponíveis e a frequência máxima de *clocks* que o sistema consegue rodar sem erros (Gafsi et al, 2021; He et al, 2021).

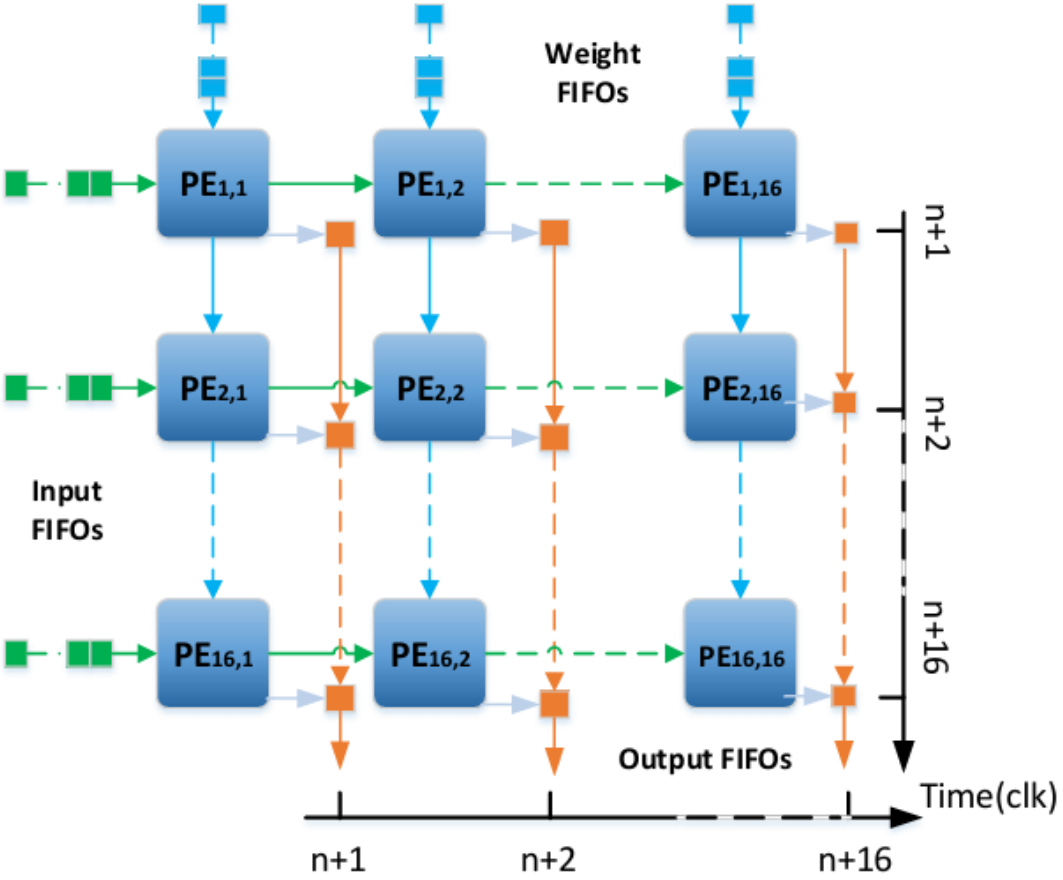

Figura 6 - Estrutura de uma Matriz Sistólica

Fonte: Gafsi et al, 2021

2.1.2.2 Módulo da função de ativação sigmoide

Após a multiplicação matricial feita no estágio anterior, o módulo da função de ativação Sigmoide lê a entrada do buffer de resultado temporário, e calcula a função de ativação ao consultar a tabela e obtém os vetores de saída i, f, o e g, como exemplificado nas Fórmulas (1)-(6). A tabela de consulta é pré-determinada na RAM, permitindo um histórico de cálculos, fazendo com que resultados já conhecidos não sejam recalculados sem necessidade. A saída deste módulo também é armazenada na memória global do chip (He et al, 2021).

2.1.2.2.1 *Função de ativação*

Um dos desafios ao utilizar redes neurais, é lidar com termos exponenciais que costumam reduzir a velocidade de processamentos dos cálculos, assim como aumentando a complexidade do algoritmo. A fim de simplificar a função de ativação não linear como a tangente hiperbólica (tanh) e a sigmoide, exibidas nas Fórmulas (1) a (6), que costumam ser utilizadas também para arranjos de redes neurais (Pogiri, Ari e Mahapatra, 2022).

Aproximação polinomial primeiro divide a função em diversos intervalos. Em cada intervalo, a fórmula da aproximação polinomial mostrada na Fórmula 7, é implementada para transformar a computação complexa não linear em uma simples computação polinomial, elimina os termos de ordem alta, e mantêm apenas os termos de baixa ordem necessários para garantir sua acurácia. Necessita de múltiplas operações para cada valor para obter um resultado, o que consome muitos recursos e ciclos *clock* (Pogiri, Ari e Mahapatra, 2022).

$$f(x) = p_o + p_1 x + p_2 x^2 + \cdots + p_n\, x^n \qquad (7)$$

O método da tabela de consulta armazena os resultados de funções não lineares previamente calculados, a fim de quando essa função se repetir, os valores possam ser retirados simples e rapidamente da tabela consulta, poupando tempo de processamento e uso de recursos (He et al, 2021).

O tamanho da tabela de consulta depende da precisão do processamento e quanto maior a precisão, maior seu tamanho. He et al, (2021) utilizou a aritmética de ponto fixo, onde um número de ponto fixo corresponde a um pequeno intervalo de ponto flutuante.

Com o tamanho da tabela de até 4096, o que é uma quantidade aceitável, pode ser até ser vista como operação de quantificação. Pogiri, Ari e Mahapatra (2022, apresentam ser possível uma redução do tamanho da tabela de consulta, explorando-se mais recursos de tanh e a função de ativação sigmoide. Também Foram observados os seguintes pontos sobre a função Tanh e a função Sigmoide tanto em He et al (2021), quanto em Pogiri, Ari e Mahapatra (2022).

- Função Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

A função tanh é simétrica em sua origem, então $\tanh(x) = -\tanh(-x)$ e com esta simetria, pode-se olhar apenas para o lado positivo do eixo X. Quando X está no intervalo (0, 0.25), o resultado é muito próximo de x, neste intervalo a função pode ser aproximada como $\tanh(x) = x$

Em resumo, a função tanh, pode ser expressa aproximadamente como a Fórmula 8 (Pogiri, Ari e Mahapatra, 2022).

$$f(x) = \begin{cases} -\tanh(-x), & x < 0 \\ x, & 0 \le x < 0.25 \\ \tanh\_table(x), & 0.25 \le x \le 3 \\ 1, & otherwise \end{cases} \quad (8)$$

- Função Sigmoide: $sigmoid(x) = \frac{1}{1+e^{-x}}$

A função sigmoide é simétrica no ponto (0, 0.5), então $sigmoid(x) = 1 - sigmoid(-x)$. Com esta simetria, apenas a metade positiva do eixo X precisa ser considerado (He et al, 2021). Em resumo, a função sigmoide pode ser expressa aproximadamente como a Fórmula 9.

$$f(x) = \begin{cases} 1 - sigmoid(-x), & x < 0 \\ sigmoid\_table(x), & 0 \le x \le 6 \\ 1, & otherwise \end{cases} \quad (9)$$

O diagrama da função ativação aproximada pelo método da tabela de consulta é mostrada na Figura 7 (Gafsi et al, 2021; He et al, 2021). Como observado, a função aproximada é quase idêntica a original. A Figura 8 mostras comparativamente os indicadores de performance referente a latência e uso de recursos entre a função aproximada e original.

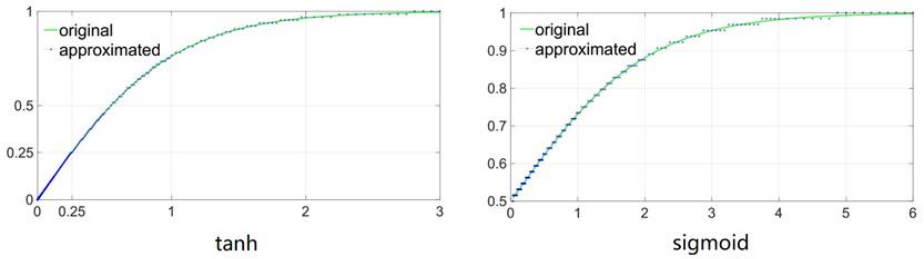

Figura 7 - Comparação gráfica entre a função de ativação aproximada com a original

Fonte: He, J. et al, 2021

Method	Function	LUT	FF	Latency (clks)
Polynomial	tanh	4287	2036	22
	sigmoid	3003	1382	16
Lookup Table	tanh	2224	1429	15
	sigmoid	1608	891	10

Figura 8 - Comparativo do uso de recurso e latência entre aproximação polinomial e o método de tabela de consulta.

Fonte: Gafsi et al, 2021; He et al 2021

2.1.2.3 Módulo *element-wise*

O último estágio consiste no módulo de processamento *element-wise* que, como podemos observar na Figura 2, lê os vetores de saída do módulo da função de ativação, i ,f ,g e o, que estão armazenados no buffer e executa operações de adição e multiplicação, gerando o vetor de estado de célula c_t e o vetor de saída h_t e os envia como input para o próximo passo. Após completar todos os passos necessários, a saída final é escrita novamente na memória do host (Kim, Cheng e Ko, 2022).

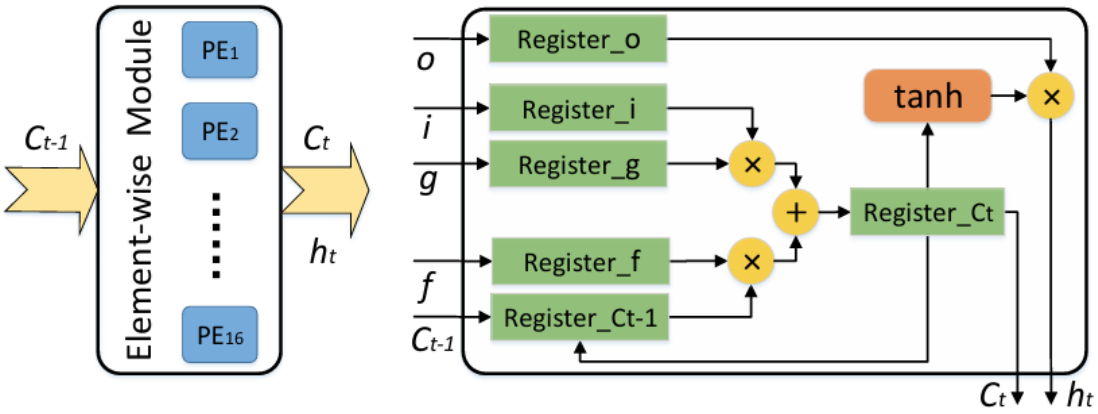

Figura 9 - Módulo Element-wise

Fonte: Gafsi et al, 2021; He et al 2021

2.1.2.4 Operação de ponto-fixo

Em circuitos de dados existem diversos tipos de dados, sendo um deles os pontos flutuantes, também conhecidos como *float*. Há diversos tipos de *float* e sua utilizações são amplas, pois permitem armazenar dados com até 16 casas decimais, no caso dos *double-precision floats*, o que permite muita precisão para as aplicações que as necessitam.

Porém, por se tratar de valores numéricos de grande volume, consomem muitos recursos de hardware e aumentam a latência de processos. A fim de contornar esse uso elevado de hardware, alta latência e mantendo um nível de precisão satisfatório, foi utilizado uma substituição de variáveis *float*, por variáveis de ponto fixo. Os detalhes dessa operação podem ser vistos em He et al (2021).

2.2 AUTENTICAÇÃO *RFID* EM DISPOSITIVOS USANDO BLOCOS DE CRIPTOGRAFIA EM PLATAFORMAS *FPGAS*

Segundo Raghavendra (2023), a tecnologia *RFID* fornece estabilidade, confidencialidade e segurança, porém, possui diversos problemas relacionados a armazenamento, arquitetura e segurança.

A fim de solucionar esses problemas, foi proposto a abordagem de integrar, um algoritmo de autenticação com criptografia *XTEA* e *HMA*, para fornecer uma camada a mais de segurança a ataques (Raghavendra, 2023).

2.2.1 Identificação por Rádio Frequência (RFID)

Os Radio-Frequency Identification Devices (RFID), são dispositivos que utilizam ondas de rádio para identificar outros dispositivos, por meio de tags, que armazenam seus dados de identificação. Possuem uma autenticação robusta, porém com baixo alcance e grau de segurança.

2.2.1.1 *Tags*

As *tags* possuem chips que armazenam dados e uma antena para transmitir os dados por meio de ondas de rádio frequência. Elas podem ser ativas (alimentadas por baterias) ou passivas (alimentadas pelo campo eletromagnético do leitor).

2.2.1.2 *Reader*

O dispositivo *reader* emite ondas de rádio que ativam a *tag* e recebe os dados transmitidos por ela.

2.2.1.3 *Middleware*

Após a comunicação é necessário armazenamento e processamento adequado dos dados coletados, e o hardware responsável por isto se chama *Middleware*.

2.2.2 *Lightweight Block Ciphers (LBC)*

Os *Lightweight Block Ciphers (LBC)* são algoritmos de criptografia projetados para sistemas com recursos limitados, como dispositivos de baixo poder de processamento ou que operam em ambientes restritos. Eles são otimizados para oferecer uma camada de criptografia em situações em que a eficiência de recursos, como memória e capacidade de processamento, são fundamentais. Dentre eles estão o *Extended Tiny Encryption Algorithm (XTEA)* e o *Hummingbird Algorithm (HMA)*, que são amplamente utilizados em dispositivos embarcados e outras aplicações.

2.2.2.1 Extended tiny encryption algorithm (XTEA)

O *Extended Tiny Encryption Algorithm (XTEA)* é uma versão aprimorada do *Tiny Encryption Algorithm (TEA)*, que provê maior segurança e performance, ao incorporar mecanismos de *pipeline*. O algoritmo é capaz de gerar textos criptografados, por meio de processo de *bit shifting* e operadores lógicos XOR. Possui uma chave de 128-bit com um tamanho de bloco de 64-bit. Possui 3 principais processos, que são: agendamento de chave, criptografia e descriptografia (Raghavendra, 2023)

Input: P_T, RF_i, RF_0, m
Output: E_T or D_T

1. Pre-processing Phase:
   - I. If $(m = 0)$, then $P_1 = P_0 = P_T[31 : 0]$ and $P_0 = P_T$ [63:32]; otherwise,
   - II. $P_1 = P_T[31 : 0]$ and $P_0 = P_T[63 : 32]$;
2. Operation of Rounding Function:
   - I. c = 0: $RF_i = P_1$;
   - II. c = 1: if $(m = 0)$ then $P_0 = P_0 + RF_0$ else $P_0 = P_0 - RF_0$;
   - III. c = 2: $RF_i = P_0$;
   - IV. c = 3: if $(m = 0)$ then $P_1 = P_1 + RF_0$ else $P_1 = P_1 - RF_0$;
3. Generation of XTEA output:
   If $(m = 0)$ then $E_T = P_1, P_0$ else $D_T = P_0, P_1$;

Figura 10 - Algoritmo do *XTEA*

Fonte: Raghavendra, 2023

2.2.2.1.1 *Agendamento de chaves*

Paralelamente aos processos de criptografia e descriptografia, acontece o agendamento de chaves. Dependendo de qual etapa do contador c e qual o modo m, o agendamento de chaves acontece a seguinte forma:

$$Enc: K_0 = K[3\&X_E] + X_E; \quad if\,(c = 0)$$
$$Enc: K_0 = K[3\&X_E >> 11] + X_E; if\,(c = 2)$$
$$Dec: K_0 = K[3\&X_D >> 11] + X_D; \quad if\,(c = 0)$$
$$Dec: K_0 = K[3\&X_D] + X_D; \quad if\,(c = 2)$$

Figura 11 - Fórmulas do agendamento de chaves. Baseado no modo e etapa do contador.

Fonte: Raghavendra, 2023

2.2.2.1.2 *Fase de pré-processamento*

É a fase na qual é atribuído o valor da chave P_T, P_0 e P_1 obterão, baseado no modo atual. Sendo $m = 0$ para modo de criptografia, e $m = 1$ para descriptografia (Raghavendra Rao P, 2023).

1. Pre-processing Phase:
   I. If $(m = 0)$, then $P_1 = P_0 = P_T[31 : 0]$ and $P_0 = P_T\,[63{:}32]$; otherwise,
   II. $P_1 = P_T[31 : 0]$ and $P_0 = P_T[63 : 32]$;

Figura 12 - Fase de pré-processamento do algoritmo *XTEA*

Fonte: Raghavendra, 2023

2.2.2.1.3 *Operação da função rodada*

A função rodada é um processo baseado na estrutura de Feistel e é a mesma para os processos de criptografia e descriptografia. Sua saída é determinada pela etapa de agendamento de chave, previamente citada, sendo descrita como:

$$RF_0 = ((RFi \gg 5) \wedge (RFi \ll 4) + RFi) \wedge K_0 \quad (10)$$

A etapa de operação da função rodada avalia em qual etapa o contador c esta, e qual o modo m atual. Se $m = 0$ criptografia, se $m \neq 0$, descriptografia. Seguindo o comportamento abaixo:

2. Operation of Rounding Function:
   I. c = 0: $RF_i = P_1$;
   II. c = 1: if $(m = 0)$ then $P_0 = P_0 + RF_0$ else $P_0 = P_0 - RF_0$;
   III. c = 2: $RF_i = P_0$;
   IV. c = 3: if $(m = 0)$ then $P_1 = P_1 + RF_0$ else $P_1 = P_1 - RF_0$;

Figura 13 - Fase da função rodada do algoritmo *XTEA*

Fonte: Raghavendra, 2023

2.2.2.2 Algoritmo de criptografia *hummingbird (HMA)*

Algoritmo de criptografia utilizado para sistemas com baixa capacidade de processamento. Devido ao seu baixo uso de processamento é adequado para soluções onde o hardware já é intensamente utilizado em outros processamentos, ou com hardwares com baixa capacidade computacional.

O algoritmo é composto por uma chave secreta de 128 bits; Uma chave 256 bits responsável por registrar o estado interno. O processo de criptografia do *HMA* é dividido nas etapas de Inicialização e Criptografia e seu fluxo está na Figura 14 (Raghavendra, 2023).

Input: Nonce Initialization: N_1, N_2, N_3, and N_4;
16-bit input Text (I_i) and 256-bit key
(composed into K_1, K_2, K_3, and K_4)
Output: E_T or D_T

- **Initialization Stage**
  1. Register (State) formation $R1$, $R2$, $R3$, and $R4$ using nonces.
     I. $R1 = N_1$;
     II. $R2 = N_2$;
     III. $R3 = N_3$;
     IV. $R4 = N_4$;
  2. Initialization of LFSR:
     I. LFSR = 16'haaaa;
- **Encryption Stage**
  1. Total Rounds r = 0, 1, 2, 3;
  2. Operation of Block Encryption (E):
     I. $T1 = E_{K1}\ (I_i \boxplus R1_r)$;
     II. $T2 = E_{K2}\ (T1_r \boxplus R2_r)$;
     III. $T3 = E_{K3}\ (T2_r \boxplus R3_r)$;
     IV. $CT_i = E_{K4}\ (T3_r \boxplus R4_r)$;
  3. Internal state Updation:
     I. $LFSR_{r+1} = LFSR_r$;
     II. $R1_{r+1} = R1_r \boxplus T3r$;
     III. $R2_{r+1} = R2_r \boxplus T1_r \boxplus R4_{r+1}$;
     IV. $R3_{r+1} = R3_r \boxplus T2_r \boxplus LFSR_{r+1}$;
     V. $R4_{r+1} = R4_r \boxplus T1_r \boxplus R1_{r+1}$;
  4. Return CT_i and loop completes

Figura 14 - *HMA* completo

Fonte: Raghavendra, 2023

2.2.2.2.1 *Estágio de inicialização*

O processo de inicialização é a fase em que ocorre a devida atribuição de valores às variáveis que utilizaremos no estágio de criptografia. Os registradores de estado são compostos por $R1$, $R2$, $R3$ e $R4$, usando *nonces.*

No processo de inicialização, $R1$ recebe o *nonce* $N1$, e assim sucessivamente. o *Linear Feedback Shift Register (LFSR)*, que é responsável por adicionar uma camada de aleatoriedade ao processo de criptografia, recebe o valor inicial de 16'haaa, que corresponde em hexa decimal o valor binário: 1010 1010 1010 1010 (Raghavendra, 2023).

Input: Nonce Initialization: N_1, N_2, N_3, and N_4;
16-bit input Text (I_i) and 256-bit key
(composed into K_1, K_2, K_3, and K_4)
Output: E_T or D_T

- **Initialization Stage**
  1. Register (State) formation $R1$, $R2$, $R3$, and $R4$ using nonces.
     I. $R1 = N_1$;
     II. $R2 = N_2$;
     III. $R3 = N_3$;
     IV. $R4 = N_4$;
  2. Initialization of LFSR:
     I. LFSR = 16'haaaa;

Figura 15 - Estágio de inicialização do *HMA*

Fonte: Raghavendra, 2023

2.2.2.2.2 *Estágio de criptografia*

O estágio de criptografia é dividido em blocos, o Bloco de Operação de Criptografia, o Bloco de Atualização de estado Interno e por fim é gerado o texto criptografado CT_i.

2.2.2.2.3 *Bloco de operação de criptografia*

Neste bloco, é representado o funcionamento do processo de criptografia do *HMA*. De acordo com a rodada r atual, o valor do output de estado segue as etapas da Figura 16. Paralelamente a cada rodada, o valor correspondente do $LFSR$ e R, são ajustados conforme a rodada atual.

- **Encryption Stage**
  1. Total Rounds r = 0, 1, 2, 3;
  2. Operation of Block Encryption (E):
     I. $T1 = E_{K1}\ (I_i \boxplus R1_r)$;
     II. $T2 = E_{K2}\ (T1_r \boxplus R2_r)$;
     III. $T3 = E_{K3}\ (T2_r \boxplus R3_r)$;
     IV. $CT_i = E_{K4}\ (T3_r \boxplus R4_r)$;
  3. Internal state Updation:
     I. $LFSR_{r+1} = LFSR_r$;
     II. $R1_{r+1} = R1_r \boxplus T3r$;
     III. $R2_{r+1} = R2_r \boxplus T1_r \boxplus R4_{r+1}$;
     IV. $R3_{r+1} = R3_r \boxplus T2_r \boxplus LFSR_{r+1}$;
     V. $R4_{r+1} = R4_r \boxplus T1_r \boxplus R1_{r+1}$;
  4. Return CT_i and loop completes

Figura 16 - Estágio de criptografia do *HMA*

Fonte: Raghavendra, 2023

3 EXEMPLO DE PROPOSTA PARA USO DE DATA CENTERS COM FPGA

A fim de potencializar a eficiência de cálculos e estudos que utilizam *Deep Learning* como *LSTM*, o uso de hardware apropriado para tal função é fundamental. Como abordado no capítulo de fundamentação teórica, é possível utilizar *FPGAs* como hardware especializado em cálculos *LSTM* conforme proposto por He et al. (2021) em ambiente de *Data Centers*.

Eles se mostram ideias para manter a segurança da comunicação da rede de dispositivos *RFID*, assim como no uso de *LBCs* para incrementar mais o baixo custo computacional e a alta eficiência energética, como proposto por He et al. (2021), Gafsi et al.(2021), Kim, Cheng e Ko (2022) e Raghavendra (2023).

A hipótese, associada ao objetivo do trabalho, se apresenta na ideia de integrar os componentes visto no capítulo da fundamentação teórica em um único ambiente, com potencial de se reduzir a um único dispositivo com FPGA, obtendo assim ambas as qualidades e propiciando um hardware completo ao que tange o uso de *LSTM*.

Por fim, após elaboração do ambiente hardware proposto, o mesmo poderia ser aplicado a um data center para efetuar e controlar os dados de consumo e previsão de demanda elétrica, dentro do munícipio do Rio de Janeiro. A ideia é que essa hipótese também possa usar o modelo para previsão de situações de alta demanda elaborado por Fogliatto et al. (2005).

3.1 PREVISÃO DE DEMANDA ELÉTRICA USANDO *LSTM*

Um dos principais desafios que se tornam evidentes junto ao acelerado crescimento tecnológico e produtivo, é a produção e distribuição de energia elétrica de forma estável e robusta. Em um país como o Brasil, com matriz energética variada, tanto de fontes intermitentes, como não intermitentes, soluções para a indústria elétrica são complexas e necessitam de tecnologia apropriada.

A previsão de demanda elétrica é um dos principais pontos que as distribuidoras enfrentam, pois devem ser capazes de fornecer energia em quantidades adequadas para a população, comércio e indústrias, de forma efetiva, sem excedentes

e sem déficit, visto que isso ocasionaria perdas financeiras e multa para a concessionária, respectivamente.

Demanda é influenciada por diversos fatores, porém ao utilizar históricos anteriores de demanda energética, com o uso de *LSTM* para processar esses dados, é possível gerar uma previsão que serve de guia para fundamentar tomadas de decisão e planejamento trianual das concessionárias.

Alguns dos fatores observáveis são o clima, sazonalidade, classe (residencial, industrial) e segmento econômico. Sendo o principal parâmetro para avaliar o consumo elétrico de um estado, a sazonalidade.

Para os cálculos de previsão de demanda elétrica propostos neste trabalho, utilizou-se dos cálculos estatísticos visto no trabalho de Fogliatto et al. (2005), que fornecem as formulas e aplicações relativas as demandas e considerações para cada etapa na criação do modelo de previsão de consumo energético de uma região.

3.2 DADOS DO EXEMPLO PROPOSTO

O controle associado a gerência sobre a demanda e consumo de energia elétrica por uma empresa, gera, por si só, uma grande quantidade de dados a serem tratados. Há desde os dados de previsão, como também os dados efetivos, e a necessidade de compara-los e quantifica-los.

Se uma empresa já gera toda uma necessidade *data center* para controle, mais empresas geraria uma necessidade maior de controle. Segundo o CCEE (2025), no ano de 2024 existiram cerca de 610.940 empresas de porte acima de média (com estabelecimento próprio) ativas, ou registradas, no Estado do Rio de Janeiro, sendo 93.000 só no Município do Rio de Janeiro.

É de conhecimento geral dos profissionais que atuam no setor elétrico, que a sazonalidade faz parte de um dos mais impactantes parâmetros que influenciam na demanda elétrica. A Figura 17, retrata a demanda geral de energia elétrica para empresas localizadas no município do Rio de Janeiro.

Pode-se observar que nos meses de dezembro, janeiro, fevereiro e março em que a temperatura é mais quente, por causa do verão e a transição para o início do outono, costuma ter maior demanda elétrica. Podemos inferir que parte do motivo para esse fenômeno, pelo menos referente ao consumo comercial, é pelo uso de equipamentos de refrigeração ou de ventilação.

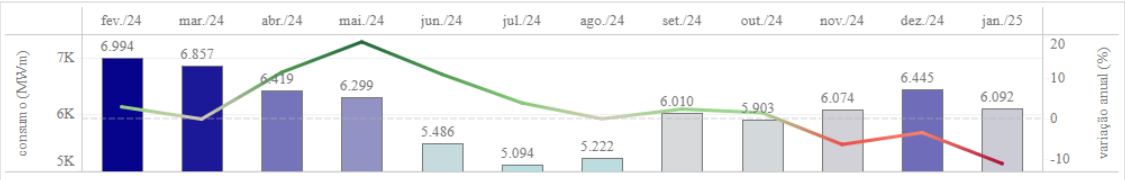

Figura 17 - Consumo Médio (MWh) e Variação do Consumo Médio (MWm). 23/02/25

Fonte: CCEE, 2025

Segundo a CCEE (2025), há como se ter acesso a base CONSUMO_HORARIO_PERFIL_AGENTE emitida pelo órgão, possibilitando o foco no consumo baseado no perfil do consumidor, data da comercialização (contendo dia, ano e hora), classe do consumidor, capacidade de carga, entre outros. Assim, é possível observar e obter dados das empresas de porte médio para cima (com estabelecimento próprio), que podem ser usados no *data center* FPGA proposto

Um dado que pode ser observado numa análise superficial, é que a demanda aumenta especialmente no período da noite, mesmo entre algumas empresas, pois muitas delas trabalham em turno a noite, fazendo com que o consumo de energia continue aumentando, isso pode ser visto também na base do CCEE (2025) “CONSUMO_HORARIO_PERFIL_AGENTE”.

Com uso destes dados a própria CCEE poderia construir um *data center* usando FPGA, que seria com baixa latência, pequeno ambiente e baixo consumo, para gerenciar os dados das empresas do município do Rio de Janeiro, inclusive contando coma localização geográfica e gerencia-los de modo a oferecer tomadas de decisão baseadas em:

- Qual região apresenta maior demanda.
- Qual região apresenta menor demanda.
- Quais as previsões futuras para cada uma dessa áreas, utilizando bases históricas.

Com isso, seria possível distribuir melhor a demanda, otimizar a geração de energia e consumo, direcionar melhor essa geração, entre outras possíveis ações oriundas dos dados diários obtidos através desse *data center*.

3.3 HIPOTESE DO DATA CENTER USANDO FPGA

Data centers tem sido um enorme ponto na estrutura global de suporte para redes de computadores especializados desempenharem grandes processamentos de dados como IA e *Deep Learning*, ou em armazenamento de dados em nuvem. Isso dá-se devido a sua estrutura que proporciona condições ideais de resfriamento, demanda elétrica e rede, além de uma equipe dedicada a monitorar a operação, para garantir seu funcionamento ininterrupto.

A máquina proposta neste trabalho restringe-se ao método de rede neural *LSTM*, que possui aplicação vasta como cálculos previsão de demanda elétrica, setor financeiro, processamento de linguagem natural *(NLP)*, entre outros. Os principais desafios de implementação de *datacenters* estão relacionados a infraestrutura e a custos de operação, majoritariamente proveniente da alta demanda energética que as máquinas exigem.

A proposta deste trabalho apresenta não só um hardware otimizado para aplicações específicas, mas também adequado para baixo consumo energético. Alinhado a esses pontos, com o hardware proposto, é possível construir datacenters capazes de processar enorme volumes de dados, contornando seu principal desafio.

3.4 ANÁLISE DOS DADOS E ALGORITMO PROPOSTO

Conforme visto anteriormente na fundamentação teórica, a máquina utilizaria do método de rede neural *LSTM* para processar os dados, em conjunto a estrutura de comunicação *RFID* com criptografia. O algoritmo proposto segue a Figura 2, que representa uma célula *LSTM*, junto com as Fórmulas (1) a (6), que representam o comportamento e processamento de cada entrada de sinal recebido pela célula assim como o a saída esperada.

Conforme visto no Capítulo 2, é possível otimizar os processos da célula *LSTM*, a fim de obter melhor desempenho, sem afetar de forma relevante, a acurácia do processo. Alinhado a um algoritmo adequado para a função proposta, também é necessária uma abordagem direcionada para segurança da operação e com baixo custo computacional.

Conforme elaborado por Raghavendra (2023), os métodos de criptografia *XTEA* e *HMA*, são adequados e proporcionam uma confiança a mais na segurança do processo.

4 CONCLUSÃO E CONSIDERAÇÕES FINAIS

Pelos tópicos abordados nesse trabalho, verificou-se a possibilidade de construção de um ambiente computacional de larga escala baseado em placas de *FPGAs* direcionados para processamentos e estudos de *Deep Learning* e capazes de trabalhar com um grande volume de dados, com baixa latência e consumo energético.

Adicionados em uma rede com uma infraestrutura de datacenter, a capacidade dessa estrutura é ampliada, suportando melhor o ritmo de coleta de dados da atualidade, tornando possível um melhor controle do alto volume de dados inferido (no caso dessa proposta, os dados de consumo de energia elétrica de todo estado do Rio de Janeiro, por ano).

Devido a não existir em fácil acesso um ambiente laboratorial, a proposta prevalece no campo da hipótese, contudo, como visto na fundamentação teórica, há todo uma razão para que tal hipótese acabe sendo comprovada.

O uso de ambiente computacional com *FPGAs* está na direção das tendências globais atuais para a coleta e tratamento de dados, desta forma, esse trabalho, apresenta a proposta de uma opção de hardware para lidar com os problemas contemporâneos ao que tange coleta e tratamento de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

CAO, SHIJIE et al. Efficient and Effective Sparse LSTM on FPGA with Bank-Balanced Sparsit. In: 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, p. 63-72, 2019.

CCEE-Câmara de Comercialização de Energia Elétrica. Dados de Consumo Energético– Município do RJ, IN: https://dadosabertos.ccee.org.br/dataset/?tags=consumodados obtidos em janeiro de 2025

CONG, J. et al **FPGA HLS Today: Successes, Challenges, and Opportunities**, In: ACM Transactions on Reconfigurable Technology and Systems, NY, USA, v 15, n 4, 2022.

CHUNG, C.; LIU, C. K.; LEE, D. H. **FPGA-based accelerator platform for big data matrix processing.** In: IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), 2015, Singapura. Anais... Singapura: IEEE, 2015. p. 221-224. DOI: 10.1109/EDSSC.2015.7285090.

FOGLIATTO, F. S.; et al. **Previsão de demanda por energia elétrica: método e aplicação.** In: XXV Encontro Nacional de Engenharia de Produção (ENEGEP 2005), Porto Alegre, RS, Brasil, 29 out. – 01 nov. 2005. Anais... Porto Alegre: ABEPRO, 2005. p. 1-18. DOI: 10.14488/1676-1901.v5i4.385.

GAFSI, M. et al. **Xilinx Zynq FPGA for Hardware Implementation of a Chaos-Based Cryptosystem for Real-Time Image Protection** In: Journal of Circuits, Systems and Computers, v 38, n 11, 2021

HE, D.; et al. **An FPGA-Based LSTM Acceleration Engine for Deep Learning** Frameworks. *Electronics* **2021**, *10*, 681. https://doi.org/10.3390/electronics10060681

HOOZEMANS, et al, "**FPGA Acceleration for Big Data Analytics: Challenges and Opportunities**," in *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 30-47, Secondquarter 2021, doi: 10.1109/MCAS.2021.3071608. keywords: {Field programmable gate arrays;Computer architecture;Standardization;Reconfigurable logic;Big Data;Throughput;Hardware},

KIM, S. Y.; CHENG, W e KO, J. H., "**Element-wise Partial Product Quantization for Efficient Deep Learning Accelerators,**" *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, Yeosu, Korea, Republic of, 2022,

KUNG. **Why systolic architectures?** Computer, 15(1), p 37–46, jan 1982.

POGIRI, R; ARI S. e. MAHAPATRA, K. K., "**Design and FPGA Implementation of the LUT based Sigmoid Function for DNN Applications**," in: *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, Warangal, India, 2022, pp. 410-413, doi: 10.1109/iSES54909.2022.00090.

RAGHAVAN, R.; PERERA, D. G. **A fast and scalable FPGA-based parallel processing architecture for K-means clustering for big data analysis.** In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2017, Victoria, BC, Canadá. Anais... Victoria: IEEE, 2017. p. 1-8. DOI: 10.1109/PACRIM.2017.8121905.

RAGHAVENDRA, R., "**Secured Authentication of RFID Devices Using Lightweight Block Ciphers on FPGA Platforms**," in *IEEE Access*, vol. 11, pp. 107472-107479, 2023, doi: 10.1109/ACCESS.2023.3320277.

ROUHANI, R. D. et al. **SSketch: An automated framework for streaming sketch-based analysis of big data on FPGA.** In: IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, 2015, Vancouver, BC, Canadá. Anais... Vancouver: IEEE, 2015. p. 187-194. DOI: 10.1109/FCCM.2015.56.

WANG *et al*., "**Acceleration of LSTM With Structured Pruning Method on FPGA**," in *IEEE Access*, vol. 7, pp. 62930-62937, 2019, doi: 10.1109/ACCESS.2019.2917312.