Proposta de Implementação Paralela do Método Naive Bayes em FPGA

Matheus T. Barbosa [*] **Wysterlânya K. P. Barros** [*]
Marcelo A. C. Fernandes [*,**]

[*] *Laboratório de Aprendizagem de Máquina e Instrumentação Inteligente, nPITI/IMD, UFRN, Natal, RN, Brasil (e-mail: mtarginoo@gmail.com, kyurybarros@gmail.com).*
[**] *Departamento de Engenharia de Computação e Automação, Universidade Federal do Rio Grande do Norte, Natal, RN, Brasil (e-mail: mfernandes@dca.ufrn.br).*

Abstract: This work proposes a hardware implementation of the Naive Bayes classifier, aiming to develop a fully parallel architecture that aims to obtain high performance in processing speed and energy consumption. The proposed hardware was developed in a Field Programmable Gate Array (FPGA) using a fixed point. All details of the developed architecture are presented, including information regarding the occupation rate of hardware resources, processing time, and energy consumption for an FPGA Stratix V 5SGXMBBR3H43C3. A comparative analysis of the model was carried out with other works of state of the art through the results obtained. It was possible to conclude that the implementation obtained a performance similar or superior to other works in the literature.

Resumo: Este trabalho propõe uma implementação em hardware do classificador Naive Bayes, tendo como objetivo o desenvolvimento de uma arquitetura totalmente paralela, que visa obter alta performance em termos de velocidade de processamento e consumo energético. O hardware proposto foi desenvolvido em *Field Programmable Gate Array* (FPGA) utilizando ponto fixo. Todos os detalhes da arquitetura desenvolvida são apresentados, incluindo informações referentes à taxa de ocupação dos recursos de hardware, tempo de processamento e consumo energético para uma FPGA Stratix V 5SGXMBBR3H43C3. Através dos resultados obtidos, foi realizada uma análise comparativa do modelo obtido com outros trabalhos do estado da arte. Com isso, foi possível concluir que a implementação obteve um desempenho similar ou superior a outros trabalhos na literatura.

Keywords: Naive Bayes, Machine learning, Fully parallel, FPGA, Hardware.

Palavras-chaves: Naive Bayes, Aprendizagem de máquina, Totalmente paralela, FPGA, Hardware.

1. INTRODUÇÃO

As técnicas de *Machine Learning* (ML) costumam ser utilizadas no processamento de grandes volumes de dados, sendo necessário um processamento intensivo que ocasiona um elevado consumo energético. Diante disso, para grandes empresas de tecnologia como Microsoft, IBM, Amazon e Google, o consumo de energia pode vir a se tornar um sério problema. Por este motivo, seus servidores e bancos de dados estão realizando mudanças a nível arquitetônico com a adoção de alternativas de baixa potência, capazes de fornecer um alto poder de processamento (Caulfield et al., 2016).

As características de processamento e consumo energético das técnicas de ML geralmente dificultam sua aplicação em campos emergentes, como Comunicação 5G, *Advanced Driver Assistance System* e bioinformática. Deste modo, paralelizar o processo de computação do algoritmo é uma solução frequentemente adotada para contornar a necessidade de alto desempenho computacional requerido no processamento de dados massivos. O uso de soluções em hardware tem se mostrado muito eficiente para auxiliar na execução desses métodos. Implementações realizadas em *Field Programmable Gate Array* (FPGA) conseguem a obtenção de ótimo desempenho com baixo consumo de energia, devido a paralelização e otimização dos algoritmos a nível de portas lógicas, sendo possível acelerar em até mil vezes o processamento das técnicas comparado às implementações convencionais em software (Silva et al., 2015; Torquato e Fernandes, 2016; Holanda e Fernandes, 2016; Silva et al., 2016; Souza e Fernandes, 2014)

O Naive Bayes (NB) é uma técnica de ML bastante utilizada para a resolução de problemas de classificação. Na literatura há alguns trabalhos que implementam o NB em hardware, buscando obter processamento em tempo real para as mais diversas aplicações, como reconhecimento facial e classificação de pacotes de redes (Chou e Chen, 2020; França et al., 2015).

[⋆] O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Neste trabalho é proposto a implementação das etapas de treinamento e inferência do método de Naive Bayes em FPGA, com uma arquitetura totalmente paralela e representação numérica em ponto fixo, visando a aceleração do algoritmo e o baixo consumo energético. Foram implementados cinco designs distintos com 64, 32, 16, 8 e 4 atributos para a etapa de inferência e três designs com 16, 8 e 4 atributos para a etapa de treinamento. Resultados de ocupação de hardware, *throughput* e consumo energético foram apresentados e comparados com outros trabalhos da literatura. Os designs apresentados foram sintetizados, testados e validados no FPGA Stratix V 5SGXMBBR3H43C3.

2. ESTADO DA ARTE

Em Meng et al. (2011), foi implementado um classificador Naive Bayes em FPGA para realizar a detecção de movimentos em uma cena, a partir da análise de vetores com atributos binários. A implementação serial da etapa de treinamento e inferência do NB foi sintetizada para o FPGA Virtex-4 XC4VLX160. A arquitetura proposta ocupou 730 *Flip Flops* (FF), 819 LUTs, 784 *slices* e 3 memórias RAM. Esses resultados foram comparados com a implementação de um bSOM também em FPGA, sendo constatado que o classificador Naive Bayes consumiu menos recursos de hardware para todos os parâmetros analisados.

No trabalho de Tzanos et al. (2019) foi realizada uma implementação do Gaussian Naive Bayes a fim de otimizar o processo de aprendizagem do algoritmo. A implementação foi efetuada utilizando *High Level Synthesis* (HLS) na placa Xilinx ZedBoard para a obtenção de um sistema heterogêneo. Os resultados obtidos consideraram um modelo de classificação com 784 atributos e 10 classes, sendo utilizado 2 mil amostras do *dataset* MNIST. Na etapa de treino, a implementação apresentou uma ocupação de hardware de 197 DSP, 56 BRAM, 37.929 LUT e 29.271 FF. Enquanto na etapa de inferência foram utilizados 93 DSP, 112 BRAM, 34.977 LUT e 31.779 FF. O tempo de execução das etapas de treinamento e inferência foram, respectivamente, $1,1$ s e $2,7$ s.

No trabalho apresentado em França et al. (2015) foram implementados três classificadores: Decision Tree, Naive Bayes e K-Nearest Neighbors. Esses métodos foram aplicados na classsificação de pacotes enviados dentro de uma rede, buscando identificar possíveis pacotes maliciosos que poderiam infectar a máquina alvo. As implementações tiveram como FPGA alvo a Cyclone IV GX FPGA. Foram implementadas seis versões do Naive Bayes, sendo três combinacionais e três sequenciais, usando notação em ponto flutuante, para 32, 16 e 10 bits em cada tipo de implementação. O trabalho apresentou dados de ocupação, processamento e consumo energético para todos os designs desenvolvidos.

Em Marsono et al. (2006) foi proposto uma arquitetura em hardware do classificador Naive Bayes para a classificação de spam em *e-mail*, usando um sistema numérico logarítmico para reduzir a complexidade computacional. Foram apresentadas quatro implementações do algoritmo, sendo três em ponto fixo com precisão de 8, 12 e 16 bits e uma em ponto flutuante, sintetizadas para o FPGA Altera Stratix EP1S10F780C5ES. Esta implementação apresenta um seletor para cada representação (ponto fixo e flutuante). O hardware foi validado a partir do *dataset SpamAssassin's public corpus*. Foram apresentados dados de tempo de processamento e ocupação de hardware para cada arquitetura.

O trabalho de Chaudhary e Sharma (2017) propõe a implementação em hardware do classificador Naive Bayes para a classificação em tempo real de expressões faciais, havendo sete classes distintas. A arquitetura foi desenvolvida com notação em ponto-fixo buscando a redução dos recursos de hardware ocupados. A implementação foi realizada utilizando o Xilinx System Generator. O sistema obtido opera a uma frequência de $241,55$ MHz. A classificação teve uma acurácia de $81,94\%$.

Wahab e Milosevic (2019) propõem um sistema em hardware para acelerar o processamento e reduzir o consumo energético associado à detecção de *malware*. A implementação foi feita em FPGA e pensada para que, eventualmente, possa ser integrada a dispositivos móveis. Para realizar a detecção dos *malwares* foram implementados três algoritmos de classificação: Regressão Logística, Naive Bayes e *Support Vector Machines*. Esses classificadores foram otimizados em relação ao consumo energético, conseguindo manter uma acurácia similar às implementações em software. A arquitetura foi desenvolvida utilizando HLS, por meio do Xilinx Vivado HLS, e sintetizada para a placa de desenvolvimento ZedBoard Zynq-720 ARM/FPGA SoC.

3. MÉTODO NAIVE BAYES

A técnica Naive Bayes é baseada no Teorema de Bayes, adotando uma forte premissa "ingênua" de que existe uma independência total entre os atributos de uma determinada amostra analisada. Esta premissa é, muitas vezes, não realista para a maior parte dos problemas de classificação no qual a técnica é empregada, entretanto, ainda assim é possível alcançar altos valores de acurácia, se comparando com outras técnicas de ML mais robustas (Zhang, 2004).

O processo de classificação da técnica NB é realizado a partir do teorema de Bayes que calcula a probabilidade de um vetor de entrada $\mathbf{x}$ pertencer a uma k-ésima classe, c_k, de um conjunto de K classes, ou seja, calcular o valor da probabilidade de $P(c_k \mid \mathbf{x})$, que pode ser expresso como

$$P(c_k \mid \mathbf{x}) = \frac{P(c_k)P(\mathbf{x} \mid c_k)}{P(\mathbf{x})} \tag{1}$$

no qual $P(c_k)$ é a probabilidade da k-ésima classe c_k acontecer, $P(\mathbf{x})$ é a probabilidade de $\mathbf{x}$ ocorrer e $P(\mathbf{x} \mid c_k)$ é probabilidade de $\mathbf{x}$ acontecer dado que c_k ocorreu. O vetor de entrada $\mathbf{x}$ é expresso como

$$\mathbf{x} = [x_1, \ldots, x_i, \ldots, x_N] \tag{2}$$

onde cada i-ésimo x_i representa um atributo independente e N corresponde à quantidade total de atributos.

É possível ignorar o denominador da fração apresentada na Equação 1 por se tratar de um termo constante, ou seja, não depende do valor de c_k e cada i-ésimo valor x_i já é apresentado. Usando a premissa de independência entre os atributos, a Equação 1 pode ser reescrita como

$$P(c_k \mid \mathbf{x}) \propto P(c_k) \prod_{i=1}^{N} P(x_i \mid c_k). \quad (3)$$

Deste modo, é obtido o modelo probabilístico do NB, entretanto, para a obtenção do classificador é necessário um mecanismo de decisão para que seja possível o processo de classificação. Esta regra de decisão pode ser obtida ao tomar classe que produz a maior probabilidade de classe para uma dada amostra de entrada, ou seja,

$$\hat{k} = arg \max_{k \in 1,..,K} P(c_k) \prod_{i=1}^{N} P(x_i \mid c_k) \quad (4)$$

onde $\hat{k}$ é a estimativa de uma k-ésima classe c_k.

4. HARDWARE PROPOSTO

A implementação foi realizada com representação em ponto fixo e utilizou 15 bits na parte fracionária. Na parte inteira, a quantidade de bits varia entre 3 e 17 bits a depender do estágio do *pipeline*, visando a otimização da síntese em termos de ocupação de hardware e aumento do *throughput*. Foram implementadas duas estruturas, uma para o treinamento e outra para a inferência do NB. As arquiteturas de hardware foram desenvolvidas baseadas na descrição do método apresentado na Seção 3.

4.1 Módulo de Treinamento

A Figura 1 apresenta o módulo de treinamento, denominado de *Training Module* (TM). O TM tem por objetivo a obtenção das probabilidades $p(\mathbf{x}|c_k)$ e $p(c_k)$, apresentadas na Equação 1. A cada n-ésimo instante de tempo, t_s, o TM recebe como entrada o vetor de atributos $\mathbf{x}(n)$, apresentado na Equação 2, e a k-ésima classe, $c_k(n)$, referente a esses atributos.

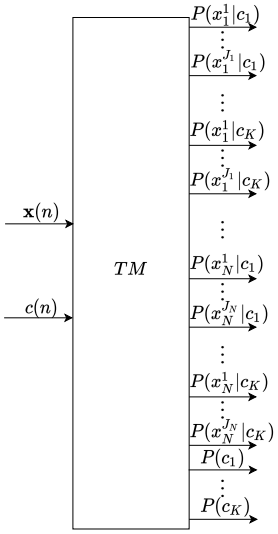

Figura 1. Visão geral do TM.

A Figura 2 detalha os módulos implementados no interior do TM, o qual é formado por um módulo para cálculo das probabilidades $P(c_k)$, denominado de *Class Probability Module* (CPM), e N módulos para cálculo das probabilidades $P(x_i|c_k)$, denominado de *Attribute Probability Module* (APM). O CPM recebe como entrada a cada n-ésimo instante um valor de $c_k(n)$ e realiza o cálculo das probabilidades $P(c_k)$, bem como o número de ocorrências de cada k-ésima classe, nc_k. Cada i-ésimo APM_i recebe como entrada a cada n-ésimo instante o i-ésimo atributo, $x_i^j(n)$, a k-ésima classe, $c_k(n)$, e o número de ocorrência de cada k-ésima classe, nc_k, calculando através desses valores as probabilidades $P(x_i^j|c_k)$. O índice j representa o j-ésimo possível valor do i-ésimo atributo e J_i a quantidade máxima de valores distintos assumidos por esse atributo.

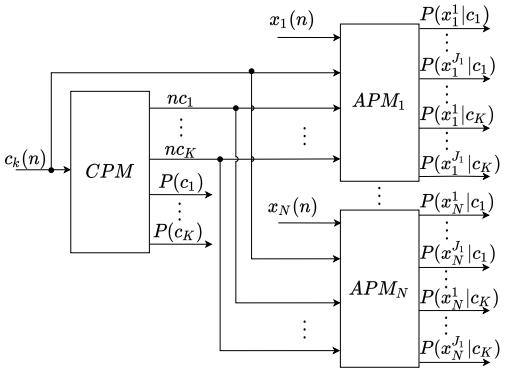

Figura 2. Arquitetura geral do TM.

Na Figura 3 é apresentada a estrutura interna do CPM. Inicialmente, cada k-ésima classe apresentada no n-ésimo instante de tempo, $c_k(n)$, é comparada com todas as possíveis classes do *dataset* $(c_1, \ldots, c_K)$, representadas na arquitetura como constantes. Caso seja verdadeira a saída do bloco comparador, o contador será habilitado e haverá o incremento com valor um para cada ocorrência da classe. Após a contabilização de todas as classes, o número de ocorrência de cada k-ésima classe, nc_k, é disponibilizado para os blocos APM_i. Posteriormente é realizada a multiplicação de cada k-ésimo nc_k por uma constante com valor de $\frac{1}{K}$ e, dessa forma, é calculada a probabilidade de ocorrência de cada k-ésima classe, $P(c_K)$, presente na Equação 3. As probabilidades calculadas são armazenados em registradores (R), e fornecidas como saída do CPM.

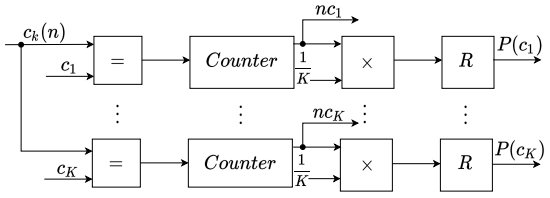

Figura 3. Arquitetura interna do CPM.

O processamento em cada APM$_\text{i}$ ocorre paralelo ao CPM. A estrutura interna de cada i-ésimo APM é apresentada na Figura 4. Cada i-ésimo APM$_\text{i}$ recebe como entrada o i-ésimo atributo, $x_i(n)$, a k-ésima classe, $c_k(n)$, e a quantidade de ocorrência de cada k-ésima classe, nc_k. Inicialmente, $x_i(n)$ é concatenado a $c_k(n)$, através do bloco representado na figura por CXU. Essa concatenação acarreta na criação de um valor $z_k^{j_i}$, que representa

a ocorrência de uma k-ésima classe dado que o i-ésimo atributo assumiu um j-ésimo valor. Através dos blocos de comparação, é detectado a ocorrência de cada $z_k^{j_i}$, sendo a saída dos blocos comparadores utilizados para habilitar um contador (*Counter*) que registra o total de ocorrência de cada $z_k^{j_i}$. A saída de cada contador é dividido por um k-ésimo nc_k relacionado ao $z_k^{j_i}$ contabilizado. Devido as limitações na implementação da divisão relacionados à obtenção dos valores decimais do resultado, foi aplicada uma multiplicação antes e depois do bloco por $\alpha_1 = 1000$ e $\alpha_2 = 0,001$. Para evitar o problema de frequência zero do Naive Bayes, o qual ocorre quando $P(x_i^j|c_k) = 0$, foram adicionadas portas lógicas "OR"após as últimas multiplicações do bloco, essas portas recebem como entrada a saída multiplicador e uma constante com o menor valor possível de ser representado na precisão utilizada, $\beta \approx 0,00031$, dessa forma, as probabilidades $P(x_i^j|c_k)$ que teriam valor igual a zero passam a assumir o valor de β. As saídas das portas lógicas são as probabilidades $P(x_i^j|c_k)$, as quais são armazenadas nos registradores (R) e fornecidas como saídas do módulo. Ao fim deste procedimento a etapa de treinamento do NB é concluída, havendo a obtenção de todos os valores de $P(x_i^j|c_k)$ e $P(C_k)$.

4.2 Módulo de Inferência

A visão geral do módulo de inferência, denominado de *Inference Module* (IM), é mostrada na Figura 5. O processo de inferência do NB consiste na classificação de uma amostra $x(n)$, conforme a Equação 4. Para que este procedimento seja possível, é necessário o cálculo da probabilidade $P(c_k|\mathbf{x})$, conforme a Equação 3, através dos dados calculados no TM.

A estrutura interna dos blocos $P(c_k|\mathbf{x})$ é apresentada na Figura 6. Conforme ilustrado, cada bloco é composto por N LUTs, configuradas com profundidade J_i, utilizando 15 bits para a representação dos valores. As LUTs tem a finalidade de relacionar cada valor de entrada, $x_i^j(n)$, a uma probabilidade correspondente, $P(x_i^j|c_k)$. As probabilidades $P(x_i^j|c_k)$ obtidas por cada i-ésima LUT são multiplicadas através de uma árvore de multiplicadores. Na última multiplicação da árvore, o resultado do produtório é multiplicado pela k-ésima probabilidade $P(c_k)$, obtendo assim o valor de $P(c_k|X)$, conforme a Equação 3.

A Equação 4 mostra que o processo de decisão da classe correspondente à amostra avaliada consiste em selecionar a classe que produz a maior probabilidade $P(c_k|\mathbf{x})$. Esse processo é realizado por um bloco de comparação que recebe como entrada o resultado de todos os blocos $P(c_k|x)$ e fornece como saída a classe c_k relacionada ao maior valor de probabilidade calculado, por meio de uma árvore de comparadores. Após a seleção da maior probabilidade da classe, é então concluído o processo de inferência.

5. VALIDAÇÃO DO HARDWARE

O hardware foi validado a partir de uma comparação dos resultados obtidos pela implementação proposta com uma implementação do NB em software, utilizando representação numérica em ponto flutuante no padrão IEEE 754. O conjunto de dados utilizados na validação foi o *divorce predictors*, Yöntem et al. (2019). Este *dataset* é composto por 170 respostas de casais (amostras) a 54 variáveis de Escala de Preditores de Divorcio (atributos) da terapia de Gottman. Os atributos possuem 5 possibilidades de resposta sobre questões intrínsecas à questões da vida conjugal, sendo elas: 0 = Nunca, 1 = Raramente, 2 = Medianamente, 3 = Frequentemente e 4 = Sempre. Esses dados são usados para a classificação de duas classes, classe 1 e 2, que representam, respectivamente, se houve ou não divórcio com base nas respostas apresentadas.

O *dataset* foi dividido em dois conjuntos de dados, um para a etapa de treinamento, correspondendo a 80% dos dados, e outro para o teste do classificador, correspondendo aos outros 20% dos dados. Os resultados da validação foram computados utilizando quatro atributos discretos, no qual cada atributo pode assumir cinco valores distintos. Deste modo, os parâmetros utilizados para a validação foram fixados em $N = 4$, $J_1 = J_2 = J_3 = J_4 = 5$ e $K = 2$.

5.1 Validação do treinamento

As Figuras 7 e 8 apresentam o erro absoluto entre os valores calculados no hardware e no software para cada k-ésima probabilidade $P(x_i^j|c_K)$, $e_{i,k}^j$, expresso como

$$e_{i,k}^j = |P_{\mathrm{H}}(x_i^j|c_k) - P_{\mathrm{S}}(x_i^j|c_k)| \tag{5}$$

onde $P_{\mathrm{H}}(x_i^j|c_k)$ e $P_{\mathrm{S}}(x_i^j|c_k)$ são os valores de probabilidade obtidos em hardware e software, respectivamente.

Além disso, também foram calculados os erros absolutos para cada k-ésima probabilidade $P(c_K)$, ec_K, dado por

$$ec_k = |P_{\mathrm{H}}(c_k) - P_{\mathrm{S}}(c_k)| \tag{6}$$

onde $P_{\mathrm{H}}(c_k)$ e $P_{\mathrm{S}}(c_k)$ correspondem, respectivamente, aos valores de probabilidade obtidos em hardware e software. O erro absoluto calculado para cada classe foi $ec_1 = 0,00303$ e $ec_2 = 0,00379$.

5.2 Validação da inferência

Para a validação da inferência foi utilizado o conjunto de dados de teste. Os dados de testes foram fornecidos como entrada para o IM e o resultado de classificação de cada amostra foi comparado com a classe esperada definida no *dataset*. A Figura 9 apresenta a matriz de confusão obtida para os resultados da inferência realizada pelo IM sobre o conjunto de testes. Conforme os resultados obtidos, o classificador obteve uma acurácia de 97,06%.

6. RESULTADOS

6.1 Resultados da proposta de hardware

Resultados da ocupação de hardware A partir da síntese da arquitetura desenvolvida para o FPGA alvo foi obtido o relatório de ocupação de recursos de hardware para as implementações do treinamento e inferência. Os dados de ocupação de hardware estão apresentados nas Tabelas 1 e 2. Em ambas as tabelas, a primeira coluna indica o número de atributos para o qual a arquitetura foi ajustada e os dados de ocupação foram obtidos ($\mathrm{N_A}$). Na segunda coluna é mostrado o número de células lógicas ocupadas no dispositivo alvo (NCL). A terceira coluna exibe a quantidade de bits de blocos de memória (NBits). E, por fim,

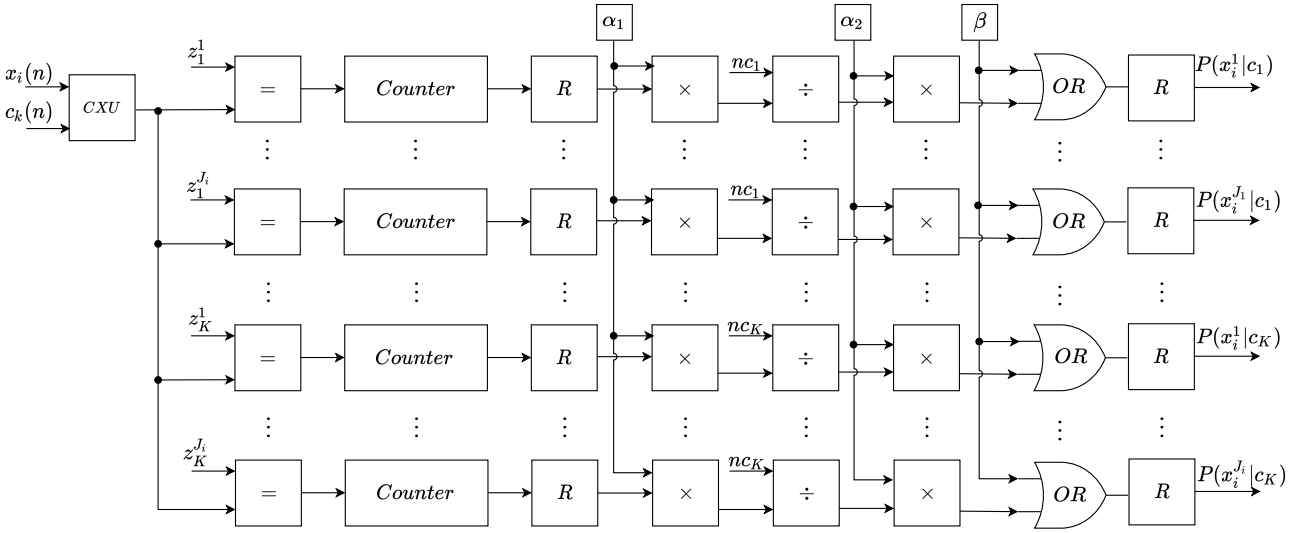

Figura 4. Arquitetura interna do i-ésimo bloco APM.

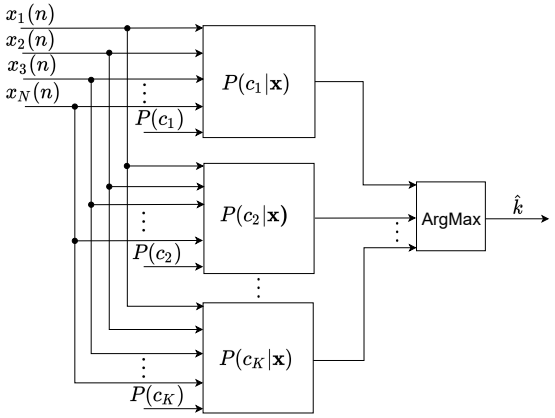

Figura 5. Visão geral do IM.

a quarta coluna apresenta o número de multiplicadores implementados utilizando blocos DSP (NMULT).

Tabela 1. Recursos utilizados no módulo de treinamento.

N_A	NCL	NBits	NMult
4	13950	2550	0
8	27825	4590	0
16	55459	8670	0

Tabela 2. Recursos utilizados no módulo de inferência.

N_A	NCL	NBits	NMult
4	24	1024	8
8	27	2048	16
16	37	4096	32
32	59	8192	64
64	121	16384	128

Resultados de tempo de processamento Os resultados do tempo de processamento foram obtidos observando a quantidade de amostras processadas por segundo na saída do bloco analisado. As Tabelas 3 e 4 mostram, respectivamente, as informações relacionadas ao tempo de processamento para os módulos de treinamento e inferência. Em ambas as tabelas, a primeira coluna indica o número de atributos para o qual a arquitetura foi ajustada e os dados de tempo foram obtidos (N_A). A segunda coluna apresenta o *throughput* em *Mega Samples Per Second* (Msps) obtido em cada implementação (THPT). A terceira coluna mostra os valores utilizados de *clock* (CLK) em cada implementação.

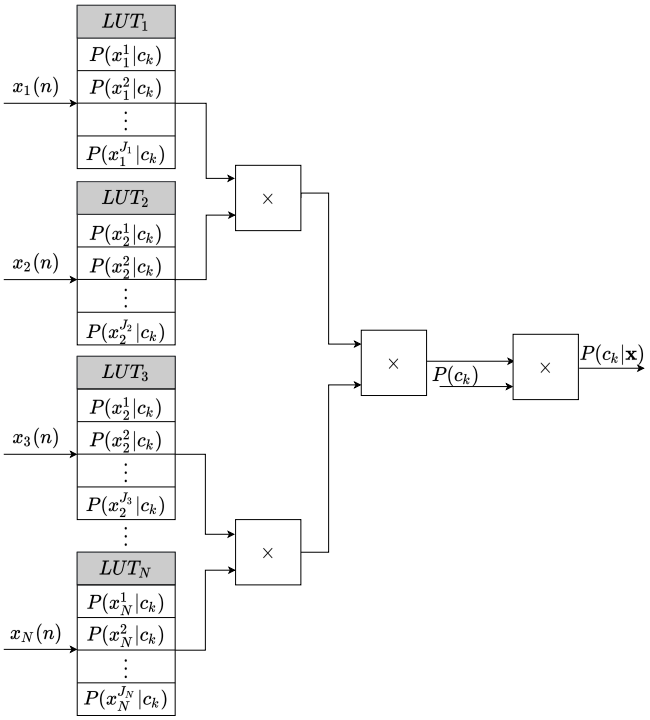

Figura 6. Arquitetura interna dos blocos $P(c_k|\mathbf{x})$

Na Tabela 3 é observado que os valores de *throughput* obtidos permanecem constantes mesmo com o aumento de N_A, sendo este resultado proveniente ao alto paralelismo aplicado no processo de treinamento do NB. De maneira

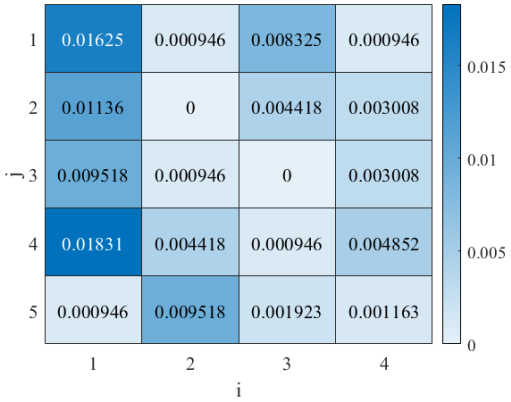

Figura 7. Valores do erro absoluto, $e^j_{i,1}$, para probabilidades $P(x^j_i|c_1)$.

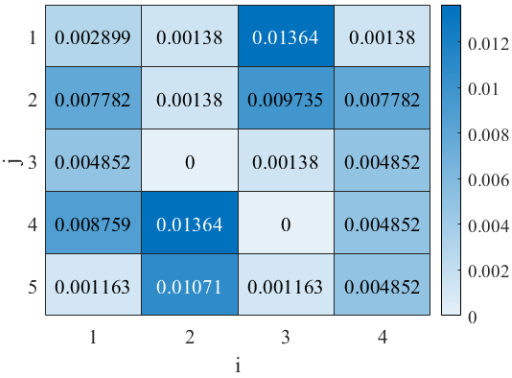

Figura 8. Valores do erro absoluto, $e^j_{i,2}$, para probabilidades $P(x^j_i|c_2)$.

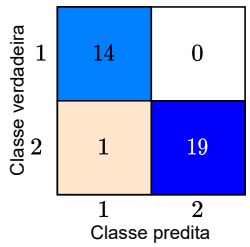

Figura 9. Matriz de confusão para as classes 1 e 2.

Tabela 3. Tempo de processamento no TM.

N_A	THPT	CLK
4	40 Msps	40 MHz
8	40 Msps	40 MHz
16	40 Msps	40 MHz

esperada, o *throughput* para o IM é reduzido com o aumento de N_A, pois ocorre o aumento na quantidade de multiplicações do processo de inferência.

6.2 Comparação com trabalhos da literatura

Para realizar as comparações da arquitetura proposta com os outros trabalhos do estado da arte, é necessário a realização de algumas conversões para que as comparações

Tabela 4. Tempo de processamento no IM.

N_A	THPT	CLK
4	72,52 Msps	75,5 MHz
8	59 Msps	59 MHz
16	50 Msps	50 MHz
32	43,01 Msps	43 MHz
64	35 Msps	35 MHz

fossem dadas em termos equivalentes de grandezas de ocupação de hardware. Isso é necessário devido diferentes placas FPGA apresentarem de forma distinta os dados de ocupação de hardware. A análise foi realizada considerando a ocupação de células lógicas e quantidade de bits no bloco de memória. As conversões foram realizadas considerando a relação entre ALMs, LUTs e Slices com seus equivalentes em número de células lógicas. Todas as conversões foram obtidas conforme a documentação oferecida pelo fabricante dos respectivos dispositivos FPGA.

A potência dinâmica, E_d, foi calculada a partir de uma aproximação que leva em consideração o clock (CLK), e os dados de ocupação de hardware, N_g, da seguinte forma

$$E_d \propto N_g \times CLK^3 \tag{7}$$

onde N_g corresponde a

$$N_g = NCL + NMult \tag{8}$$

A comparação dos valores obtidos foi realizado apenas para o módulo de inferência, devido à escassez de dados sobre a etapa de treinamento nos artigos do estado da arte. Para realizar essa comparação, foram obtidas equações a partir dos dados de síntese do módulo de inferência. Através da Tabela 2, obteve-se por regressão linear as seguintes equações relacionadas à ocupação de hardware

$$NCL = 1,6333 \times N_A + 13,0833, \tag{9}$$

$$NBits = 256 \times N_A \tag{10}$$

e

$$NMult = 2 \times N_A. \tag{11}$$

E através da Tabela 4, obteve-se por ajuste logarítmico a seguinte equação relacionada ao *throughput*

$$THPT = \frac{1}{3.5869 \times \log_2 N_A + 6,1680}. \tag{12}$$

A Figura 10 apresenta o gráfico obtido para o tempo de processamento.

Por meio das equações apresentadas, foi possível estimar valores de ocupação de hardware, *throughput* e consumo de potência dinâmica para diferentes números de atributos e, dessa forma, possibilitar a comparação de desempenho com outros trabalhos.

Comparação de ocupação do hardware Na Tabela 5, são comparados os dados de ocupação do hardware, sendo apresentado em sequência a referência que está sendo comparada, o número de atributos da referência (N_A), o número de células lógicas da referência (NCL^{Ref}), o número de células lógica deste trabalho considerando o valor de N_A da referência (NCL^{Nosso}), o ganho deste trabalho em relação à referência para o NCL (NCL^{Ganho}), o número de bits de bloco de memória da referência ($NBits^{Ref}$), o número de bits de bits de bloco de memória estimado para este trabalho considerando o valor de N_A ($NBits^{Nosso}$) e o ganho deste trabalho em relação em relação à referência para o NBits ($NBits^{Ganho}$).

Tabela 5. Comparação de recursos de hardware

Referência	N_A	NCL^{Ref}	NCL^{Nosso}	NCL^{Ganho}	$NBits^{Ref}$	$NBits^{Nosso}$	$NBits^{Ganho}$
Meng et al. (2011)	762	81	1258	$\approx 0,6510\times$	48	19.507	$\approx 0,002\times$
França et al. (2015) I	10	11.488	29,4	$\approx 390,75\times$	0	2.560	–
França et al. (2015) II	10	5.480	29,4	$\approx 186,40\times$	0	2.560	–
França et al. (2015) III	10	2.867	29,4	$\approx 97,52\times$	0	2.560	–
França et al. (2015) IV	10	1.733	29,4	$\approx 58,95\times$	10.112	2.560	$\approx 3,95\times$
França et al. (2015) V	10	1.000	29,4	$\approx 34,01\times$	6.656	2.560	$\approx 2,60\times$
França et al. (2015) VI	10	705	29,4	$\approx 23,98\times$	5.120	2.560	$\approx 2,00\times$
Marsono et al. (2006) I	15	146	37,6	$\approx 3,88\times$	38.656	3.840	$\approx 10,07\times$
Marsono et al. (2006) II	15	278	37,6	$\approx 7,39\times$	81.920	3.840	$\approx 21,33\times$
Marsono et al. (2006) III	15	313	37,6	$\approx 8,32\times$	135.168	3.840	$\approx 35,20\times$
Marsono et al. (2006) IV	15	144	37,6	$\approx 3,83\times$	135.168	3.840	$\approx 35,20\times$

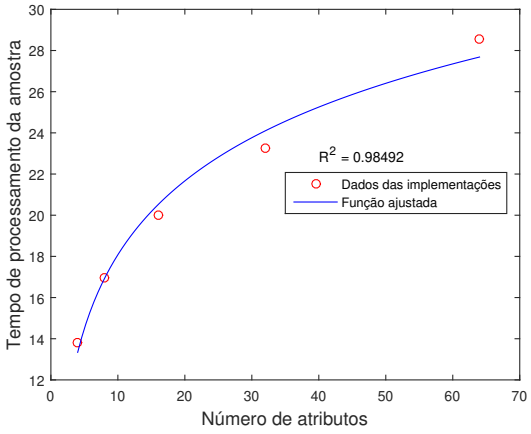

Figura 10. Função do *troughput* ajustada por uma função logarítmica.

Este trabalho conseguiu ganhos em relação ao número de células lógicas (NLC^{Ganho}) e número de bits do bloco de memória ($NBits^{Ganho}$), com exceção da arquitetura apresentada por Meng et al. (2011). Esse resultado é esperado, dado que esta implementação faz uso de uma representação em ponto fixo, enquanto França et al. (2015) faz uso de representação em ponto flutuante e Marsono et al. (2006) implementa um esquema de seleção entre ponto fixo e flutuante. Não foi possível comparar os ganhos com relação ao número de bits do bloco de memória dos designs I, II e III de França et al. (2015) por não se aplicar ao trabalho desenvolvido e, portanto, estes valores não constam na referência comparada.

Comparação de velocidade Na Tabela 6 são apresentados dados de comparação do tempo de processamento, sendo indicado em sequência a referência que está sendo comparada, o número de atributos da referência (N_A), o *throughput* apresentado pela referência ($THPT^{Ref}$), o *throughput* estimado deste trabalho considerando o valor de N_A ($THPT^{Nosso}$) e o *speedup* obtido por esse trabalho em relação à referência.

A arquitetura aqui proposta obteve resultados de *speedup* satisfatórios, alcançando desempenho superior à maioria dos trabalhos. O *throughput* alcançado foi inferior apenas aos resultados alcançados pelas arquiteturas propostas em Wahab e Milosevic (2019). Em relação aos demais trabalhos, foram obtidos valores de *speedup* de $5\times$ a $10^3\times$. Este resultado é esperado devido a arquitetura em árvore implementada pelo MI aqui proposto, o que proporciona maior velocidade de processamento da técnica.

Comparação de consumo energético Na Tabela 7 são apresentados dados de comparação do consumo energético, sendo indicado em sequência a referência comparada, o número de atributos da referência (N_A), a frequência de clock da referência (CLK^{Ref}), a frequência de clock deste trabalho para obter o mesmo *throughput* da referência (CLK^{Nosso}), a potência dinâmica consumida pela referência (CNS^{Ref}), potência dinâmica consumida por esse trabalho (CNS^{Nosso}) e a potência dinâmica economizada por esse trabalho em relação à referência comparada (PWR-Save). Para obter o mesmo *throughput* da referência o CLK^{Nosso} foi definido com valor igual a $THPT^{Ref}$. Os valores de consumo da potência dinâmica foram estimados através da Equação 7.

O trabalho Marsono et al. (2006) não utilizou multiplicadores implementados com blocos DSP e França et al. (2015) apresentou multiplicadores de tamanho 18 x 18 bits. Tamanho igual ao dos multiplicadores apresentados neste trabalho, possibilitando a comparação direta dessas quantidades com ambos os trabalhos.

A partir dos dados apresentados pode ser observado que o consumo de energia da arquitetura proposta foi menor do que todos os designs analisados, obtendo valores de economia de energia entre $10^3\times$ a $10^7\times$. Esse resultado é alcançado devido a arquitetura proposta conseguir obter elevados valores de *throughput* com uma baixa frequência de *clock* e à quantidade reduzida de recursos de hardware.

7. CONCLUSÕES

Esse trabalho apresentou uma implementação em hardware da técnica de Naive Bayes. A implementação foi desenvolvida com uma arquitetura totalmente paralela e representação em ponto fixo. As etapas de treinamento e inferência foram validadas através da comparação dos valores calculados com os de uma implementação em software utilizando ponto flutuante. Resultados de síntese foram obtidos para um FPGA Stratix V 5SGXMBBR3H43C3. Através da comparação com outros trabalhos da literatura, a arquitetura proposta obteve um *speedup* de até $10^3\times$, uma ocupação de hardware até $35\times$ menor para o número de bits e até $390\times$ menor para a quantidade de células lógicas e uma economia de energia de até $10^7\times$. Dessa forma, foi alcançado o objetivo de aceleração da técnica em hardware com redução da ocupação de recursos de hardware e consumo energético.

Tabela 6. Comparação do tempo de processamento

Referência	N_A	$THPT^{Ref}$	$THPT^{Nosso}$	Speedup
Tzanos et al. (2019)	784	$7{,}41\times 10^{-4}$ Msps	24,59 Msps	$\approx 33.200\times$
França et al. (2015) I	10	$2,8 Msps$	$54,67 Msps$	$\approx 19,53\times$
França et al. (2015) II	10	$4,16 Msps$	$54,67 Msps$	$\approx 13,14\times$
França et al. (2015) III	10	$5,51 Msps$	$54,67 Msps$	$\approx 9,92\times$
França et al. (2015) IV	10	$0,46 Msps$	$54,67 Msps$	$\approx 118,85\times$
França et al. (2015) V	10	$0,77 Msps$	$54,67 Msps$	$\approx 71\times$
França et al. (2015) VI	10	$0,98 Msps$	$54,67 Msps$	$\approx 55,79\times$
Marsono et al. (2006) I	15	$9,76 Msps$	$49,55 Msps$	$\approx 5,08\times$
Marsono et al. (2006) II	15	$8,05 Msps$	$49,55 Msps$	$\approx 6,16\times$
Marsono et al. (2006) III	15	$7,33 Msps$	$49,55 Msps$	$\approx 6,67\times$
Marsono et al. (2006) IV	15	$8,89 Msps$	$49,55 Msps$	$\approx 5,57\times$
Wahab e Milosevic (2019) I	7	$418 Msps$	$61,58 Msps$	$\approx 0,15\times$
Wahab e Milosevic (2019) II	7	$2105,39 Msps$	$61,58 Msps$	$\approx 0,03\times$

Tabela 7. Comparação do consumo energético

Referência	N_A	CLK^{Ref}	$N_g{}^{Ref}$	CNS^{Ref}	CLK^{Nosso}	$N_g{}^{Nosso}$	CNS^{Nosso}	$PWRSave$
França et al. (2015) IV	10	33,46 MHz	1747	$6{,}54 \times 10^7$	0,46 MHz	49,4	4,81	$\approx 1{,}36\times 10^7$
França et al. (2015) V	10	56,12 MHz	1004	$1{,}77 \times 10^8$	0,77 MHz	49,4	22,55	$\approx 7{,}87\times 10^6$
França et al. (2015) VI	10	71,70 MHz	707	$2{,}61 \times 10^8$	0,98 MHz	49,4	46,49	$\approx 5{,}60\times 10^6$
Marsono et al. (2006) I	15	156,18 MHz	146	$5{,}56 \times 10^8$	9,76 MHz	67,6	$6{,}28 \times 10^4$	$\approx 8{,}86\times 10^3$
Marsono et al. (2006) II	15	128,78 MHz	278	$5{,}94 \times 10^8$	8,05 MHz	67,6	$3{,}53 \times 10^3$	$\approx 1{,}68\times 10^5$
Marsono et al. (2006) III	15	117,26 MHz	313	$5{,}05 \times 10^8$	7,33 MHz	67,6	$2{,}66 \times 10^4$	$\approx 1{,}90\times 10^4$
Marsono et al. (2006)IV	15	142,23 MHz	144	$4{,}14 \times 10^8$	8,89 MHz	67,6	$4{,}75 \times 10^4$	$\approx 8{,}72\times 10^3$

AGRADECIMENTOS

O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

REFERÊNCIAS

Caulfield, A. et al. (2016). A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1–13. doi: 10.1109/MICRO.2016.7783710.

Chaudhary, P. e Sharma, M. (2017). Vlsi hardware architecture of real time pattern classification using naive bayes classifier. *ICMSSP: International Conference on Multimedia Systems and Signal Processing*, 61–65. doi: 10.1145/314511.3145521.

Chou, K. e Chen, Y. (2020). Real-time and low-memory multi-faces detection system design with naive bayes classifier implemented on fpga. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(11), 4380–4389. doi:10.1109/TCSVT.2019.2955926.

França, A. et al. (2015). The energy cost of network security: A hardware vs. software comparison. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 81–84. doi:10.1109/ISCAS.2015.7168575.

Holanda, D. e Fernandes, M. (2016). Implementação em fpga de máquina de vetores de suporte (svm) para classificação e regressão. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*.

Marsono, M., Watheq El-Kharashi, M., e Gebali, F. (2006). Binary lns-based naive bayes hardware classifier for spam control. In *2006 IEEE International Symposium on Circuits and Systems*, 4 pp.–3677. doi:10.1109/ISCAS.2006.1693424.

Meng, H. et al. (2011). Fpga implementation of naive bayes classifier for visual object recognition. In *CVPR 2011 WORKSHOPS*, 123–128. doi:10.1109/CVPRW.2011.5981831.

Silva, A. et al. (2015). Adaptive boolean logic using ferroelectrics capacitors as basic units of artificial neurons and its implementation in fpga. *Integrated Ferroelectrics*, 159(1), 23–33. doi:10.1080/10584587.2015.1029408.

Silva, L., Torquato, M., e Fernandes, M. (2016). Proposta de arquitetura em hardware para fpga da técnica q-learning de aprendizagem por reforço. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*.

Souza, A. e Fernandes, M. (2014). Proposal for parallel fixed point implementation of a radial basis function network in an fpga. In *2014 IX Southern Conference on Programmable Logic (SPL)*, 1–6. doi:10.1109/SPL.2014.7002204.

Torquato, M. e Fernandes, M. (2016). Proposta de implementação paralela de algoritmo genético em fpga. In *XXI Congresso Brasileiro de Automática (CBA 2016)*.

Tzanos, G., Kachris, C., e Soudris, D. (2019). Hardware acceleration on gaussian naive bayes machine learning algorithm. In *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 1–5. doi:10.1109/MOCAST.2019.8741875.

Wahab, M. e Milosevic, J. (2019). Power & perfomance optimized hardware classifiers for efficient on-device malware detection. *ICMSSP: International Conference on Multimedia Systems and Signal Processing*, 23–26. doi:10.1145/3304080.3304085.

Yöntem, M. et al. (2019). Divorce prediction using correlation based feature selection and artificial neural networks. *Nevşehir Hacı Bektaş Veli Üniversitesi SBE Dergisi*, 9, 259 – 273.

Zhang, H. (2004). The optimality of naïve bayes. In *In FLAIRS2004 conference*.