KCGRA – Uma Arquitetura Reconfigurável de Domínio Específico para K-means

Matheus da Silva Alves[1], **Lucas Bragança Silva**[1], **Jerônimo Penha**[1], **Ricardo Ferreira**[1], **José Augusto M. Nacif**[1]

[1]Universidade Federal de Viçosa (UFV), Viçosa, MG, Brasil

{matheus.s.alves,lucas.braganca,ricardo,jnacif}@ufv.br

Resumo. *Este trabalho apresenta um novo acelerador reconfigurável de domínio específico para o algoritmo K-Means. O acelerador pode ser totalmente reconfigurado em menos de 2,34 milissegundos para explorar diversos valores de agrupamentos e atributos e alcança até 668 Gop/s de desempenho. O acelerador foi validado em FPGAs de alto desempenho com memórias HBM. A reconfiguração dinâmica não reduz o desempenho em comparação com as implementações estáticas em HLS e RTL, que demandam horas para serem reconfiguradas. O acelerador foi construído usando um gerador genérico de CGRA parametrizado e possui um formato intermediário para sua reconfiguração de forma transparente.*

1. Introdução

O aprendizado não supervisionado oferece uma abordagem que permite extrair informações de conjuntos de dados sem a necessidade de rótulos pré-existentes, permitindo a identificação de padrões complexos e desconhecidos. Neste contexto, o algoritmo K-Means se destaca como uma técnica para análise de agrupamentos. Um de seus desafios reside na seleção dos atributos relevantes, representado neste trabalho pelo valor "N", e a determinação do número de agrupamentos, representado pelo valor "K". A tarefa de seleção dos valores de K e N pode ser custosa do ponto de vista computacional e também tem impacto direto na interpretação dos resultados e a eficácia do agrupamento. O K-means é utilizado em larga escala em aplicações de mineração de dados [Choi and So 2014]. A cada iteração, todos os pontos são avaliados. Várias iterações são executadas até alcançar a convergênia. Este processo exige esforço computacional para grandes volumes de dados [Abdelrahman 2016]. O algoritmo K-means possui potencial para a paralelização de operações com reuso de dados, que é uma característica apropriada para a aceleradores em FPGA [Penha et al. 2018, Gorgin et al. 2022].

Para acelerar a execução do algoritmo K-means, vários trabalhos apresentaram implementações com GPU [Li et al. 2013] e FPGAs [Dias et al. 2020, Paulino et al. 2020, Jain et al. 2020]. Os FPGAs podem ser mais vantajosos em termos de eficiência energética. Entretanto, as soluções em FPGAs necessitam conhecer os valores de K e N em tempo de projeto ou compilação. Mesmo usando linguagens de alto nível como C++, High Level Synthesis (HLS) [Jain et al. 2020] e OpenCL [Paulino et al. 2020] para reduzir o tempo de projeto, o tempo de compilação ainda é um grande obstáculo, pois requer horas para gerar um novo acelerador quando modificamos os valores de K e N. Este trabalho apresenta uma nova abordagem para o desenvolvimento de aceleradores em FPGA para o algoritmo K-means. O acelerador foi construído com uma camada virtual reconfigurável

sobre um FPGA, na qual os valores de K e N podem ser reconfigurados sem a necessidade de recompilar o projeto. Este recurso permite a exploração de diversos valores de K e N em tempo de execução, importantes no contexto de aprendizado de máquina, onde os pesquisadores têm como um dos principais objetivos determinar estes valores.

Dentre os diversos trabalhos com arquiteturas reconfiguráveis para aceleradores com K-means, apenas a arquitetura Versat [Lopes et al. 2017] propõe um *Coarse-Grained Reconfigurable Array* (CGRA) que possibilita a reconfiguração em tempo de execução. Entretanto, o Versat foi apenas avaliado em ambiente de simulação. Sua arquitetura possui apenas 15 elementos de processamento composta por 4 memórias de 2.048x32 bits, 6 ALUs, 4 multiplicadores e um registrador de deslocamento. A implementação do K-means pode ser feita para qualquer valor de K e N, desde de que $K \cdot N \leq 1.024$ como por exemplo $K = 32$ e $N = 32$ ou $K = 64$ e $N = 16$. Entretanto, o Versat requer $K \cdot N$ ciclos para classificar cada elemento de entrada. Além disso, utiliza uma rede crossbar completa para interligar seus 15 elementos de processamento, que inviabiliza sua escalabilidade. Diferente da arquitetura Versat, este trabalho propõe um novo CGRA reconfigurável dinamicamente para o K-means com escalabilidade e execução em pipeline, capaz de classificar um elemento de entrada em um único ciclo de *clock*.

Arquiteturas de domínio específico são boas alternativas para alcançar alto desempenho. Neste trabalho, empregamos um gerador de código aberto para arquiteturas heterogêneas de CGRA conforme proposto por [Silva et al. 2020], a fim de criar uma arquitetura dedicada ao algoritmo K-means. Este trabalho apresenta quatro contribuições principais. Em primeiro lugar, desenvolvemos dois elementos de processamento especializados: um redutor de distância e um filtro de seleção de agrupamento. Ambos os elementos podem ser ajustados dinamicamente com base nos valores de K e N, o que aumenta a flexibilidade da arquitetura. A segunda contribuição é mostrar que a nova arquitetura tem desempenho de até 668 Gops/s, sendo competitiva em relação às arquiteturas estáticas. A terceira contribuição é a utilização de um formato intermediário independente do FPGA alvo para reconfiguração dinâmica. A quarta contribuição é o tempo de reconfiguração de apenas 2,34ms em relação às arquiteturas estáticas que requerem de 2 a 4 horas para serem compiladas. Finalmente, a proposta foi validada em FPGAs Xilinx U55C com memórias HBM e todos os experimentos foram realizados considerando tempo de execução, incluindo entrada e saída de dados em comparação com vários trabalhos que fazem apenas estimativas com simulações.

O artigo está organizado da seguinte forma. A Seção 2 apresenta brevemente o algoritmo K-means. A Seção 3 descreve a nova arquitetura, enquanto a Seção 4 exibe os experimentos para avaliar a proposta. Por fim, as Seções 5 e 6 discutem os trabalhos relacionados e as principais conclusões.

2. Algoritmo K-means

O K-means é uma técnica de agrupamento não supervisionada que categoriza amostras em grupos. O hiperparâmetro K define o número de centroides ou classes. Uma amostra é atribuída a um grupo cujo centroide é o mais próximo. No entanto, o algoritmo avalia todos os pontos a cada iteração, podendo levar várias iterações para convergir, especialmente com grandes volumes de dados [Abdelrahman 2016] onde se pode explorar os FPGAs [Gorgin et al. 2022]. Embora a geração do agrupamento seja um problema NP-difícil, várias

heurísticas iterativas foram propostas [Lloyd 1982]. As heurísticas percorrem e classificam todas as amostras. Em um segundo passo, os centroides são ajustados e o algoritmo é executado novamente até convergir. Este artigo avalia o passo de classificação cuja a complexidade é $O(KNM)$ para M amostras com N atributos em K grupos.

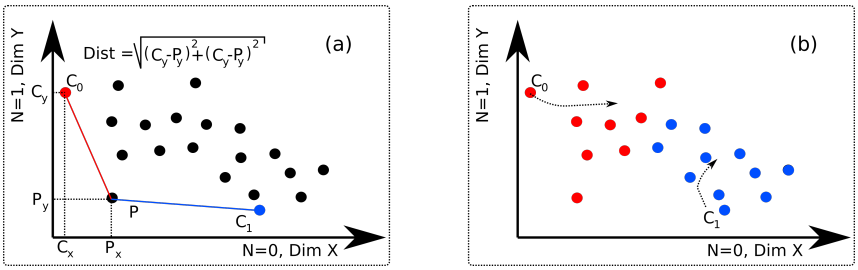

Figura 1. Exemplo de K-means com k=2, n=2: (a) Dados sem rótulos, (b) Dados agrupados nos centroides C_0 e C_1.

A Figura 1 apresenta um exemplo de K-means que possui 2 centroides C_0 e C_1, ou seja, K=2 e duas dimensões X e Y com $N = 2$. Na Figura 1(a) mostramos os dados sem rótulos, onde o ponto P está sendo classificado em função das distâncias de C_0 e C_1. Todos os dados são avaliados pelo algoritmo e suas distâncias comparadas em relação a C_0 e C_1. Se o ponto estiver mais próximo de C_0 (ou C_1) será classificado no primeiro ou segundo agrupamento, mostrado usando as cores azul e vermelho na Figura 1(b).

3. Arquitetura de Domínio Específico: KCGRA

As arquiteturas CGRAs (Arquiteturas Reconfiguráveis de Grão Grosso) podem ser otimizadas para uma aplicação específica, resultando em melhor desempenho, já que a arquitetura, além de reconfigurável, pode ter seus operadores projetados para executar diretamente os principais cálculos da aplicação alvo. Entretanto, existem poucos CGRAs comerciais. A arquitetura Versat é um exemplo de CGRA acadêmico que foi apenas simulado [Lopes et al. 2017]. A construção de CGRAs com uma camada virtual em um FPGA permite a prototipação e reconfiguração dinâmica. O ambiente HPCGRA [Silva et al. 2020] é um gerador de código aberto para arquiteturas heterogêneas de CGRA sobre FPGAs comerciais, capaz de gerar código Verilog a partir de uma descrição JSON básica. Nossa primeira contribuição para o gerador HPCGRA foi uma extensão para trabalhar com elementos de processamento mais complexos que permitam múltiplas entradas e saídas.

Além disso, neste trabalho, foram desenvolvidos dois operadores específicos para acelerar a execução do K-means. Eles encapsulam uma série de operações básicas em um único elemento de processamento, fazendo uso das outras funcionalidades do HPCGRA para interconectá-los com as memórias externas e o processador, além de oferecer um modo transparente de reconfiguração com um formato intermediário. O primeiro operador específico criado realiza a classificação em relação a dois centroides para um ponto de entrada P com até 32 atributos, ou seja, $N = 32$ e $K = 2$. Este operador foi denominado Kmeans2x32. O segundo operador específico, denominado por Filtro, faz a junção/redução de dois ou mais operadores do Kmeans2x32. Por exemplo, podemos fazer uma árvore com três operadores filtros para agrupar 4 operadores Kmeans2x32 para gerar um Kmeans8x32.

A Figura 2(a) apresenta o diagrama geral da arquitetura do KCGRA onde os operadores em vermelho são do tipo K-means 2xN e os operadores em azul do tipo filtro.

O acelerador pode ter até N entradas $I_0, I_1, \ldots, I_{N-1}$ que são os atributos do ponto a ser classificado. Cada operador 2xN com dois centroides seleciona o centroide mais próximo. Nos próximos níveis teremos uma árvore de redução com operadores do tipo filtro que irão fazer a seleção do centroide final mais próximo.

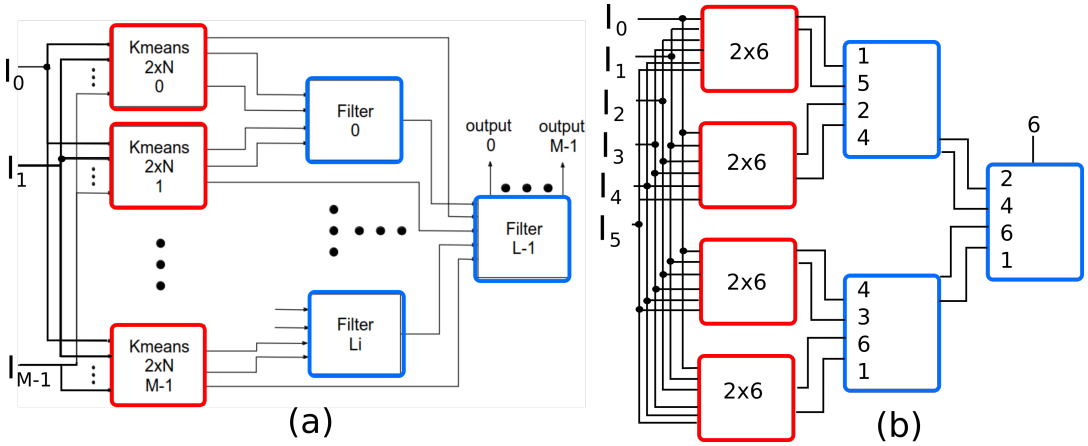

Figura 2. (a) Arquitetura KCGRA; (b) Exemplo $K = 8, N = 6$.

A Figura 2(b) apresenta um exemplo com 8 centroides ($K = 8$) e 6 atributos ou dimensões. Cada operador 2x6 tem 2 saídas que informam qual é o centroide mais próximo e qual é o valor da distância. Suponha que os centroides estejam distribuídos de cima para baixo na Figura 2(b) com o primeiro operador com C_0 e C_1, depois C_2 e C_3, C_4 e C_5 e finalmente C_6 e C_7 no último. Neste exemplo, os centroides C_1, C_2, C_4 e C_6 foram selecionados com as distâncias $5, 4, 3$ e 1, respectivamente. O primeiro filtro escolhe entre C_1 e C_2, como C_2 está mais próximo com a distância 4 é propagado para o próximo filtro. O mesmo ocorre com C_6 com distância 1. Como C_6 está mais próximo que C_4, a saída tem o valor 6 como resposta. Todos os cálculos são executados em pipeline e a cada ciclo de *clock* temos um ponto classificado.

3.1. Operadores K-means

A Figura 3(a) mostra uma visão geral do operador K-means 2xN. O elemento de processamento pode ter até 32 vizinhos para receber os atributos de entrada. Internamente o operador tem dois centroides com N atributos cada e um identificador ID para cada centroide. Desse modo, o operador irá fazer o cálculo de todas as distâncias, somar e retornar qual dos dois centroides é o mais próximo. O operador se ajusta podendo ser configurado para um valor i entre 1 e 32 para o número de atributos. Além disso, pode ser configurado para calcular 1 ou 2 centroides. A saída do operador informa a menor distância e o ID do centroide mais próximo. Estes dados serão repassados para os operadores filtro para finalizar a classificação.

A Figura 3(b) mostra o operador filtro com mais detalhes. Ele receberá dados de 2 operadores K-means. Estes dados são comparados e repassados o índice da menor distância e o ID do centroide mais próximo. Os operadores filtro podem ser ligados para formar uma árvore binária para fazer uma redução dos K centroides.

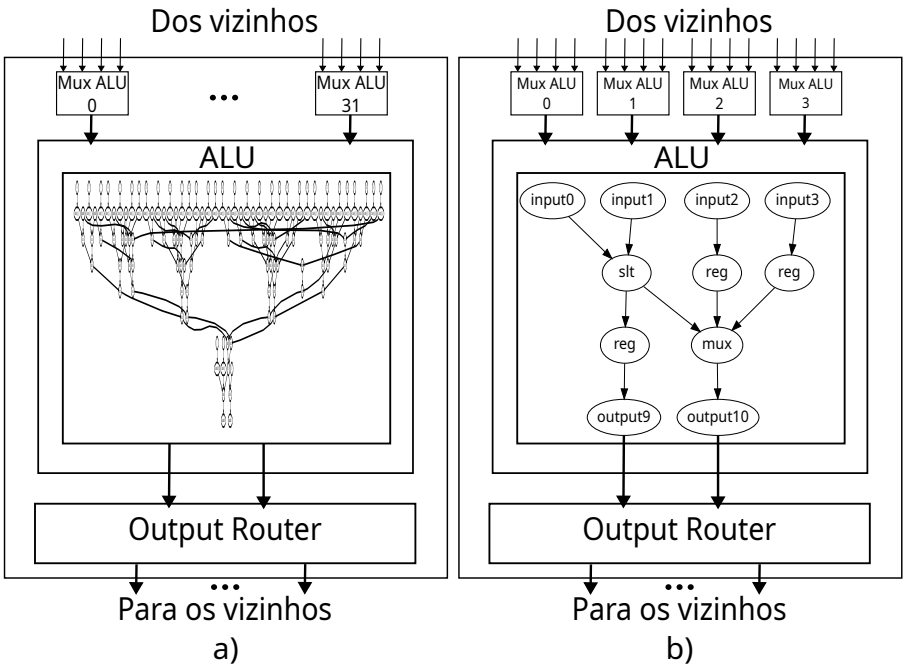

Figura 3. Arquitetura dos elementos de processamento do KCGRA.

3.2. Modo de Reconfiguração do KCGRA

Além do modo básico de configuração introduzido na seção anterior, o KCGRA permite outros modos de configuração para se adaptar a vários valores de K e N, gerando replicações do acelerador que podem classificar mais de um ponto por ciclo. Por exemplo, é possível ter dois classificadores 2x4 que irão processar dois pontos por ciclo, receberão 8 entradas (4 atributos de cada ponto) e irão gerar duas saídas com a classificação dos 2 pontos.

A Figura 4 mostra 2 exemplos de configuração das operações K-means. Suponha um operador 2x4. Para implementar um K-means 2x2 poderíamos apenas subutilizar um operador 2x4, configurando $N = 2$. A Figura 4(a) mostra que é possível configurar o operador 2x4 para receber dois pontos com $N = 2$ mas processar apenas um e calcular o centroide mais próximo, destacado em cinza. Vemos também que os dois pontos estão conectados em um segundo operador 2x4, em preto, para selecionar o centroide do segundo ponto. Portanto, implementamos o cálculo de um K-means 2x2 com o processamento de 2 pontos por ciclo. A Figura 4(b) mostra o grafo das operações que estão encapsuladas em um operador 2x2. As Figuras 4(c) e (d) mostram a reconfiguração para calcular um K-means 4x2. Neste caso os operadores são subutilizados, pois recebem apenas dois atributos e cada operador calcula dois centroides que depois são enviados a um operador filtro.

O gerador HPCGRA [Silva et al. 2020] é reconfigurado por meio de um formato intermediário ou um *assembly* CGRA. Este formato foi estendido neste trabalho para trabalhar com elementos de processamentos com múltiplas saídas e entradas. A Figura 5 apresenta o *assembly* KCGRA para a geração da configuração para os dois exemplos da Figura 4. A Figura 5(a) mostra em duas colunas a configuração para dois $K = 2, N = 2$ ou 2x2 da Figura 4(a-b) e as duas colunas de código da Figura 5(b) correspondem a configuração do $K = 4, N = 2$ ou 4x2 da Figura 4(c-d).

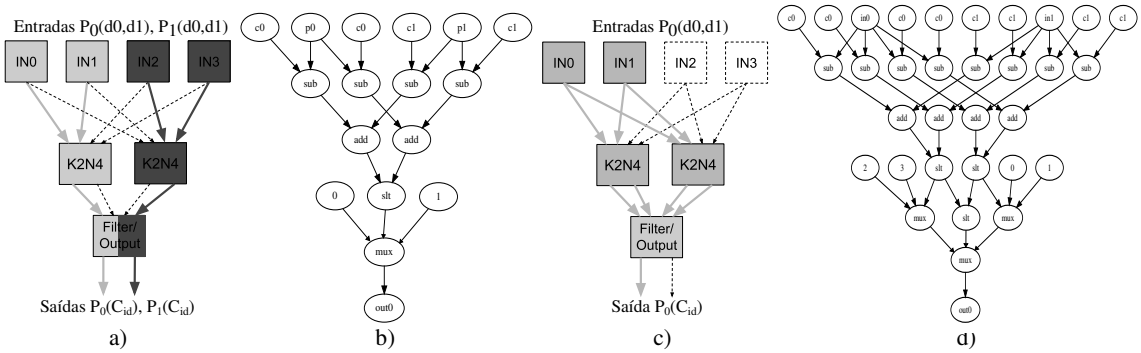

Figura 4. KCGRA com 4 entradas e operadores $K = 2$ e $N = 4$: (a) duas cópias em paralelo para K=2 e N=2; (b) Grafo de fluxo de dados com $K = 2$ e $N = 4$; (c) Configuração para $K = 4$ e $N = 2$. (d) Grafo de fluxo de dados para $K = 4$ e $N = 2$

```
00 pass $0 $istream[0]
01 pass $1 $istream[0]
02 route $0 $alu[0] $4
03 route $1 $alu[0] $4
04 Kmeans2x4 $4 $0 $1 0 0
05 route $4 $alu[1] $6
06 pass $6 $4
07 route $6 alu[0] $10
08 route $10 $6 $ostream[0]
09 pass $2 $istream[0]
10 pass $3 $istream[0]
11 route $2 $alu[0] $5
12 route $3 $alu[0] $5
13 Kmeans2x4 $5 $2 $3 0 0
14 route $5 $alu[1] $8
15 pass $8 $5
16 route $8 alu[0] $10
17 route $10 $8 $ostream[1]
```

a)

```
00 pass $0 $istream[0]
01 pass $1 $istream[0]
02 route $0 $alu[0] $4
03 route $1 $alu[0] $4
04 route $0 $alu[0] $5
05 route $1 $alu[0] $5
06 Kmeans2x4 $4 $0 $1 0 0
07 Kmeans2x4 $5 $0 $1 0 0
08 route $4 $alu[0] $6
09 route $4 $alu[1] $7
10 route $5 $alu[0] $8
11 route $5 $alu[1] $9
12 pass $6 $4
13 pass $7 $4
14 pass $8 $5
15 pass $9 $5
16 route $6 alu[0] $10
17 route $7 alu[0] $10
18 route $8 alu[0] $10
19 route $9 alu[0] $10
20 Kmeans_filter $10 $6 $8 $7 $9
21 route $10 $alu[1] $ostream[0]
```

b)

Figura 5. Código Intermediário: (a) Configuraração com duas cópias do K-means K=2 e N=2(Figura 4(a-b)); (b) Configuração para Figura 4(c-d) com K=4 e N=2.

Neste exemplo temos 4 instruções: $pass, route, Kmeans2x4, Kmeans\_filter$. Na Figura 5(a) tem quatro instruções $pass$ nas linhas 0, 1, 9 e 10, que irão receber as 4 entradas (duas para cada ponto) e irão atribuir aos elementos de processamento (PE) 0, 1, 2 e 3, respectivamente. O identificador do PE é precedido pelo caractere $. Depois quatro instruções de $route$ são usadas para conectar duas entradas em cada um dos dois operadores $Kmeans2x4$ identificados pelos ID 4 e 5, respectivamente. Na linha 4, o novo operador $Kmeans2x4$ é configurado internamente para processar os dados das entradas, onde os PE_0 e PE_1 foram conectados às entradas 0 e 1 da unidade de cálculo do operador $Kmeans2x4$. O mesmo acontece na linha 13 para os PEs de entrada 2 e 3. Finalmente, as linhas 5-8 e 14-17 fazem a ligação da saída 1 da ALU (que tem o ID do centroide) para repassar os centroides dos dois pontos para as saídas externas do KCGRA.

A Figura 5(b) mostra o segundo exemplo no qual temos apenas um ponto de entrada por ciclo com 2 atributos e calculamos 4 centroides. Neste exemplo, temos apenas dois atributos de entrada e 4 centroides. Observe que só as linhas 0 e 1 têm instruções para leitura na memória. Nas linhas 6 e 7 temos dois operadores $Kmeans2x4$ que irão receber os dados e cada um verifica 2 centroides. Depois os resultados de 2 pares de valores, a distância e o ID do centroide de menor distância, totalizando 4 valores são repassados para o operador $Kmeans\_filter$ da linha 20. Por isso, temos 4 instruções de repasse e roteamento nas linhas 12 a 19.

4. Resultados

Criamos um KCGRA com até 32 entradas, 16 operadores $Kmeans2x32$ e 15 operadores $Kmeans\_filtro$ para validar nossa abordagem do acelerador reconfigurável de domínio específico. O KCGRA foi descrito no formato do HPCGRA [Silva et al. 2020] com as extensões que adicionamos neste trabalho. Para validação e medida do tempo de execução em operação, o projeto gerado foi sintetizado e testado em uma placa aceleradora com FPGA Alveo U55C conectada em um nó cluster de alto desempenho.

O principal objetivo dos experimentos é a verificação se o desempenho do KCGRA é equivalente ao desempenho de aceleradores estáticos gerados em tempo de projeto ou compilação. O KCGRA é reconfigurado dinamicamente, sendo que a virtualização do KCGRA em comparação com uma arquitetura específica estática não gera perda de desempenho e apresenta, como grande vantagem, o ajuste dos valores de K e N em tempo de execução, sem a necessidade de recompilar, que demanda horas nas abordagens estáticas. Além disso, o KCGRA comprova que o gerador HPCGRA [Silva et al. 2020], genérico e código aberto de CGRA, pode ser aplicado e estendido para o desenvolvimento de aceleradores de domínio específicos.

Todos os experimentos foram realizados em um nó cluster com Linux Ubuntu 16.04, dois processadores Intel(R) Xeon(R) Silver 4210R, totalizando 20 núcleos, 14 MB de cache L3 e 2,4GHz de *clock* com um FPGA Xilinx Alveo U55C com 1.304K de LUTs, 2.607K registradores e 16G de memória HBM (*High Bandwidth Memory*). Os experimentos executaram os algoritmos K-means com 2 milhões de pontos de entrada e o desempenho foi comparado com as abordagens estáticas [Bragança et al. 2021, Silva et al. 2022] executando no mesmo *hardware*.

Diferente da maioria dos trabalhos com FPGA que apenas apresenta estimativas do tempo de execução com simulações, todas as medidas de tempo foram realizadas em código para a execução completa da classificação dos 2 milhões de pontos, incluindo tempos de transferência de entrada e saída de dados. Além da métrica de tempo de execução, usaremos a métrica de Giga Operações por Segundo (Gops/s) para avaliar o desempenho. Como o K-means é limitado pela vazão de memória para leitura de pontos, podemos observar que para um determinado valor de N, o tempo de execução não aumenta quando incrementamos o valor de K, pois o acelerador faz reuso dos dados. Em compensação, podemos observar o aumento do desempenho medidos em Gops/s.

A Tabela 1 apresenta uma comparação dos tempos de execução de uma iteração da fase de classificação do K-means de duas abordagens estáticas (HLS e RTL) e da abordagem KCGRA. A abordagem HLS utiliza um código C++ otimizado por meio de diretivas de compilação HLS. O código está disponível no conjunto de *benchmark* Rodinia HLS [Cong and et al. 2018]. A abordagem *Register Transfer Level (RTL)* é um acelerador especializado para o K-means [Bragança et al. 2021] que foi encapsulado com um arcabouço em Python para simplificar a chamada e geração do acelerador com integração no Jupyter Notebook [Silva et al. 2022].

O KCGRA com operadores $Kmeans2x32$ pode ser configurado para vários valores de K e N. Os testes da Tabela 1 executaram três configurações de K com 4, 8 e 16 centroides para um valor fixo de $N = 8$. Portanto, o KCGRA teve uma perda de desempenho na execução com 16x8, devido a uma subutilização na transferência de dados,

Tabela 1. Classificação de 2 milhões de pontos. Tempo de execução medidos com *time.time()* do Python3 em ms [Silva et al. 2022] e para o KCGRA usando *std::chrono::high_resolution_clock* no código da execução. Aceleradores: RodiniaHLS otimizado [Cong and et al. 2018], RTL [Bragança et al. 2021] e KCGRA.

Acelerador	K	N	CPU→FPGA	FPGA→CPU	Tempo Kernel	Acel. Kernel	Tempo Total	Acel. Total
HLS U55C	4	8	5,89	0,84	7,20	1	39,28	1
	8	8	6,26	0,92	7,20	1	47,79	1
	16	8	7,27	1,05	7,36	1	49,61	1
RTL U55C	4	8	2,93	0,37	3,27	2,20	19,74	1,99
	8	8	2,84	0,33	3,44	2,09	19,59	2,44
	16	8	2,85	0,37	3,30	2,23	19,35	2,56
KCGRA U55C	4	8	4,04	0,55	2,32	3,11	9,63	4,07
	8	8	3,54	0,54	2,32	3,10	9,14	5,23
	16	8	7,16	1,11	4.59	1,60	18,34	2,70

uma vez que o KCGRA foi previamente projetado para o valor de $N = 32$. A última coluna mostra a aceleração normalizada tendo como base o tempo da versão HLS [Cong and et al. 2018]. Para as configurações 4x8 e 8x8, o KCGRA tem um desempenho melhor que as versões estáticas, sendo 5 e 2,5 vezes mais rápido que os aceleradores HLS e RTL estático, considerando o tempo total que inclui as transferências de dados entre a CPU e o FPGA. O KCGRA calcula 4 pontos em paralelo nas configurações 4x8 e 8x8.

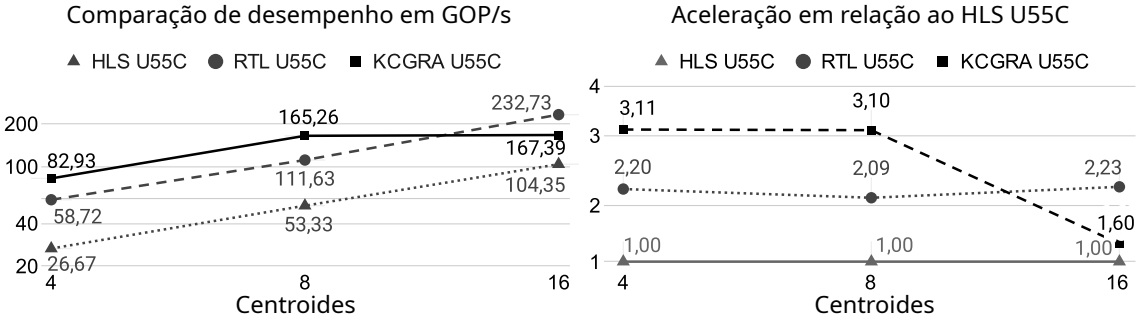

Figura 6. Comparativo de desempenho em Gops/s para a classificação da versão RTL e do KCGRA em uma Alveo U55C, para diferentes valores K e N.

A Figura 6 apresenta dois gráficos do ganho de desempenho do KCGRA em relação às abordagens RTL e HLS considerando agora o tempo de execução do kernel apenas, sem considerar a transferência de dados. A Figura 6 (a) mostra a comparação em Gops/s. É possível observar que o KCGRA é superior ao HLS nos três casos de teste e só fica abaixo da abordagem RTL para o Kmeans16x8, devido à subutilização da arquitetura. Esta subutilização ocorre pois o operador tem 32 entradas ($N = 32$) e foi configurado para $N = 8$ com $K = 16$, gerando uma subutilização das entradas. A Figura 6 (b) apresenta a aceleração das abordagens RTL e KCGRA em relação ao HLS, mostrando um ganho de até 3 vezes do KCGRA sobre o HLS para execução do kernel, enquanto o RTL apresentou um ganho de 2x.

O segundo experimento mostra uma comparação do desempenho em Gops/s do KCGRA executando no FPGA Alveo U55C em comparação com o acelerador estático

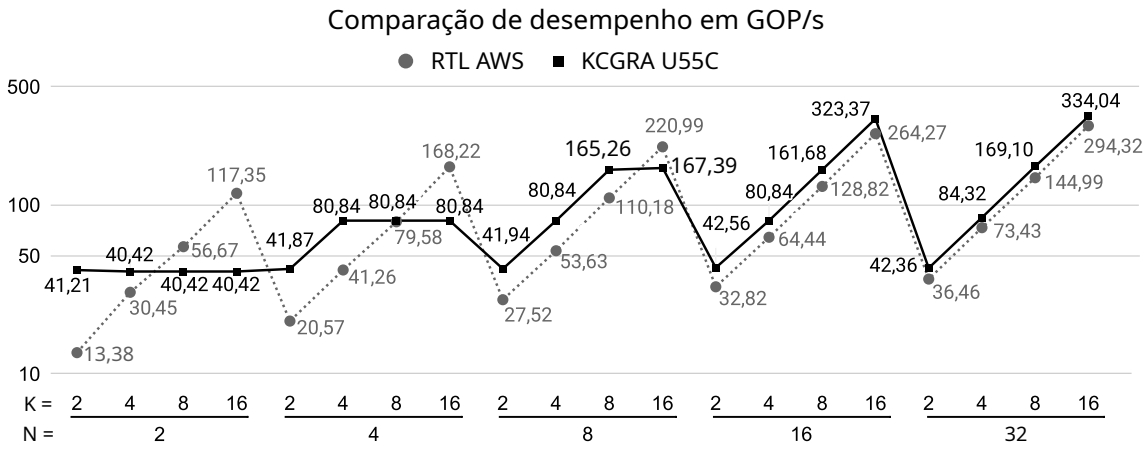

Figura 7. Comparativo de desempenho para a classificação com K-means em RTL executado na AWS FPGA e na arquitetura KCGRA executado na placa Alveo U55C, para diferentes números de centroides(K) e dimensões(N) em Gops/s.

personalizado [Bragança et al. 2021] executando no FPGA VU9P UltraScale+ na nuvem AWS da Amazon. A Figura 7 apresenta 20 configurações diferentes para N variando de 2 a 16, onde para cada valor de N, o valor de K varia de 2 a 16. O desempenho na plataforma AWS segue um padrão de uma onda dente-de-serra. Para um dado valor de N, devido ao reuso do dado, o valor em Gops/s aumenta em função de K. Para valores baixos de N, o KCGRA 2x32 apresenta desempenho baixo, pois a arquitetura aloca 32 canais de entrada e saída de dados que são subutilizadas para valores baixos de N. As duas implementações apresentam um padrão similar onde a diferença de desempenho está no acoplamento com a entrada/saída de dados.Entretanto, para $N = 8, 16$ ou 32, o desempenho do KCGRA supera o desempenho da implementação estática.

Outro ponto importante é o uso de recursos no FPGA. A Tabela 2 apresenta os resultados do uso de recursos dos aceleradores apresentados em [Silva et al. 2022] e da abordagem KCGRA. Os aceleradores foram sintetizados para a mesma plataforma, permitindo uma comparação direta dos valores de LUTs (unidades lógicas), REGs (registradores), LUTasMEM (unidades lógicas configuradas como memórias), BRAM (blocos distribuídos de memória) e DSP (unidade de cálculo no nível de palavra ou ALUs). Como já mencionado, os aceleradores em HLS e RTL são implementações que possuem os parâmetros do K-means constantes no número de dimensões e grupos, diferente do KCGRA, que possui como diferencial a reconfigurabilidade em tempo de execução sem a necessidade de realizar uma nova síntese. Assim, ao gerar um acelerador com valores menores de K e N será alocado uma quantidade personalizada e menor de recursos. Como esperado, o KCGRA apresenta um maior uso de recursos de LUTs, LUTasMEM e REGs, uma vez que são alocados recursos para sua configuração máxima ao se criar 16 unidades com operador $Kmeans2x32$ e 15 unidades com operador $Kmeans\_filtro$. O KCGRA pode implementar um K-means com 32 centroides de 32 atributos, ou seja, 32x32 em comparação com as versões estáticas que apresentam uma configuração máxima de 16x8. Ou seja, o KCGRA suporta um K-means 8 vezes maior com o dobro de centroides e o quádruplo de atributos. A configuração 32x32 foi executada para 2M pontos com um desempenho de 668 Gop/s.

Enquanto as implementações HLS e RTL fazem uso de módulos BRAM e DSP

Tabela 2. Comparação utilização de recursos para aceleradores com valores de K e N fixos e KCGRA com K até 32 e N até 32 para o FPGA Alveo U55C.

Acelerador	K	N	LUT	LUTasMEM	REG	BRAM	DSP	Reconfiguração
HLS	4	8	8.241	1.581	14.152	119	96	03h 05m 45s
	8	8	10.501	1.467	16.637	119	192	03h 10m 21s
	16	8	16.579	2.253	22.617	119	384	03h 11m 10s
RTL	4	8	9.350	306	11381	7	384	02h 44m 25s
	8	8	17.632	318	21.768	7	768	02h 56m 28s
	16	8	34.361	402	40.041	7	1.536	03h 23m 53s
KCGRA	2..32	2..32	153.494	6298	158497	0	0	2,34ms

para o cálculo das distâncias, a versão do KCGRA não usa DSPs. O cálculo da distância no operador $Kmeans2x32$, por escolha de projeto, utiliza a distância de "Manhattan" que foi sintetizada usando LUTs, uma escolha automática da ferramenta de síntese.

Por fim, a coluna **reconfiguração** mostra o tempo para gerar a configuração para programação do FPGA. Enquanto as versões estáticas precisam recompilar todo o projeto que podem demorar horas. O KCGRA, no pior cenário, para configurar todas as unidades requer 2.34ms. O tempo do KCGRA só depende do tamanho da configuração e estimamos o pior caso. Ou seja, o KCGRA ocupa um espaço compatível com as suas funcionalidades, gera um desempenho competitivo com as versões estáticas e pode ser configurado para outros valores de K e N em apenas 2,34ms, ou seja, 6 ordens de grandeza mais rápido.

5. Trabalhos relacionados

Apesar de vários aceleradores em GPU e FPGA terem sido propostos para o K-means, apenas um trabalho com reconfiguração dinâmica em um CGRA foi proposto para a arquitetura Versat [Lopes et al. 2017]. A arquitetura Versat é genérica com um CGRA com 15 elementos de processamento simples: memória, multiplicador, ALU, ou deslocador. Semelhante ao KCGRA, a arquitetura Versat possui um formato intermediário em um assembly específico para o CGRA. O acelerador do K-means foi implementado usando 691 instruções do Assembly Versat. Entretanto, o k-means foi apenas simulado para 10 mil pontos, em contraste com a nossa implementação que executou a classificação de 2 milhões de pontos. Além disso, o Versat requer $K \cdot N$ ciclos para classificar um ponto, enquanto que o KCGRA executa um ou mais pontos por ciclo. O KCGRA foi validado em um nó de processamento com FPGA em um cluster, já o Versat foi apenas simulado.

A implementação do algoritmo K-means utilizando OpenCL, uma linguagem de alto nível para FPGA, foi apresentada no trabalho [Tang and Khalid 2016]. Entretanto, o desempenho não passou de 50 Gops/s. Apenas para valores muito grande de K, maiores que 128, porém em geral, as aplicações de K-means buscam valores pequenos de K com poucos grupos. Com a evolução do OpenCL para FPGA, um trabalho mais recente [Paulino et al. 2020] apresenta dez estilos diferentes de descrição do algoritmo K-means em OpenCL. Entretanto, além de ser uma implementação estática, só foi avaliada para conjuntos pequenos de dados com 4.096 elementos considerando $k = 2$ e $N = 4$.

Outra abordagem é o projeto de geradores de aceleradores em RTL. O trabalho apresentando em [Dias et al. 2020] descreve um gerador com módulos para calcular distância e módulos para selecionar o centroide mais próximo, que produz código VHDL que foi simulado em um FPGA Virtex-6. O desempenho máximo aferido foi de 53 milhões

de pontos por segundo. O KCGRA, proposto aqui, também explora o paralelismo espacial e em pipeline, além de ser capaz de produzir um resultado por ciclo, ou seja, atingir o desempenho de 300 milhões de pontos por segundo, além de ser uma abordagem dinâmica.

O KCGRA dá continuidade a trabalhos anteriores [Penha et al. 2018, Bragança et al. 2021]. O trabalho inicial [Penha et al. 2018] mostrou um gerador para K-means em GPU e FPGA, que foi validado na plataforma HARP de FPGA da Intel. O gerador foi melhorado em [Bragança et al. 2021] e avaliado também na plataforma Xilinx na Amazon, onde os resultados apresentaram ganhos de tempo de execução de até 1,98 vezes. Posteriormente, para simplificar o uso do gerador, seu acoplamento com *notebooks* computacionais como Jupyter e o uso de Python em FPGA com memória HBM foram apresentados no trabalho [Silva et al. 2022]. Os resultados mostraram um desempenho de até 24x em comparação a um sistema com duas Xeon(R) 4210R com 10 núcleos cada, mesmo considerando a transferência de dados. Entretanto, estes trabalhos não eram reconfiguráveis dinamicamente. Portanto, para cada valor de K e N temos que recompilar. A Tabela 3 resume as abordagens relacionadas.

Tabela 3. Resumo das abordagens propostas

Trabalho	Dinâmico	HLS/RTL	Validação	Exec.	Gops/s	Pontos
[Lopes et al. 2017]	Sim	-	Simulação	Não	-	10 mil
[Tang and Khalid 2016]	Não	OpenCL	Simulação	Não	150	2 milhões
[Paulino et al. 2020]	Não	OpenCL	Simulação	-	50	4 mil
[Penha et al. 2018]	Não	Gerador RTL	Execução	Sim	40	2 milhões
[Dias et al. 2020]	Não	RTL	Simulação	Não	4	4 mil
[Bragança et al. 2021]	Não	Gerador RTL	Execução	Sim	220	2 milhões
[Gorgin et al. 2022]	Não	RTL	Simulação	Não	-	-
Este trabalho	Sim	JSON HPCGRA	Execução	Sim	668	2 milhões

6. Conclusão

O K-means tem se destacado no contexto de aprendizado não supervisionado. Porém, enfrenta desafios, como a seleção dos parâmetros "N" e "K", que influenciam tanto a eficácia do agrupamento quanto a interpretação dos resultados. Este estudo introduz uma nova abordagem para o desenvolvimento de aceleradores K-means, projetado como uma camada virtual reconfigurável sobre o FPGA, permitindo a reconfiguração dinâmica de de "K" e "N". A proposta incorpora uma nova arquitetura de domínio específico fazendo a extensão de um gerador CGRA heterogêneo de código aberto [Silva et al. 2020]. As duas principais contribuições deste trabalho são: (a) a criação de elementos de processamento especializados e ajustáveis dinamicamente; (b) a demonstração do desempenho competitivo da nova arquitetura, alcançando até 668 Gops/s. A arquitetura foi validada empiricamente em FPGAs Xilinx U55C, em comparação com outros aceleradores para K-means que foram apenas simulados. Trabalhos futuros incluem o desenvolvimento de outros operadores no ambiente HPCGRA [Silva et al. 2020] para outras aplicações em aprendizado de máquina.

7. Agradecimentos

FAPEMIG (APQ-01577-22), CNPq, Funarbe. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) Código de Financiamento 001.

Referências

Abdelrahman, T. S. (2016). Accelerating k-means clustering on a tightly-coupled processor-fpga heterogeneous system. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 176–181.

Bragança, L., Canesche, M., Penha, J., Carvalho, W., Comarela, G., Nacif, J. A. M., and Ferreira, R. (2021). An open source custom k-means generator for aws cloud fpga accelerators. In *Brazilian Symposium on Computing Systems Engineering (SBESC)*.

Choi, Y.-M. and So, H. K.-H. (2014). Map-reduce processing of k-means algorithm with fpga-accelerated computer cluster. In *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE.

Cong, J. and et al. (2018). Understanding performance differences of fpgas and gpus. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 93–96. IEEE.

Dias, L. A., Ferreira, J. C., and Fernandes, M. A. (2020). Parallel implementation of k-means algorithm on fpga. *IEEE Access*, 8:41071–41084.

Gorgin, S., Javaheri, D., and Lee, J.-A. (2022). An energy-efficient k-means clustering fpga accelerator via most-significant digit first arithmetic. In *IEEE ICFPT*.

Jain, A., Goel, P., Aggarwal, S., Fell, A., and Anand, S. (2020). Symmetric k-means for deep neural network compression and hardware acceleration on fpgas. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):737–749.

Li, Y., Zhao, K., Chu, X., and Liu, J. (2013). Speeding up k-means algorithm by gpus. *Journal of Computer and System Sciences*, 79(2):216–229.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

Lopes, J. D., de Sousa, J. T., Neto, H., and Véstias, M. (2017). K-means clustering on cgra. In *International Conference on Field Programmable Logic and Applications (FPL)*.

Paulino, N., Ferreira, J. C., and Cardoso, J. M. (2020). Optimizing opencl code for performance on fpga: k-means case study with integer data sets. *IEEE Access*.

Penha, J., Bragança, L., Coelho, K., Canesche, M., Nacif, J. A. M., and Ferreira, R. (2018). A gpu/fpga-based k-means clustering using a parameterized code generator. In *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*.

Silva, L., Canesche, M., Ferreira, R., and Nacif, J. A. (2020). Hpcgra - an orthogonal designed cgra generator for high performance spatial accelerators. In *XXI Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*.

Silva, L., Penha, J., Ribeiro, D., Silva, A., Nacif, J., and Ferreira, R. (2022). Hpyc-fpga - integração de aceleradores em fpga de alto desempenho com python para jupyter notebooks. In *XXIII Simpósio em Sistemas Computacionais de Alto Desempenho*.

Tang, Q. Y. and Khalid, M. A. (2016). Acceleration of k-means algorithm using altera sdk for opencl. *ACM Trans on Reconfigurable Technology and Systems (TRETS)*, 10(1).