Projeto e Avaliação de uma Arquitetura do Algoritmo de Clusterização *K-means* em VHDL e FPGA

Lucas Andrade Maciel, Matheus Alcântara Souza, Henrique Cota de Freitas

Grupo de Arquitetura de Computadores e Processamento Paralelo (CArT)
Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte, Brasil

{lamaciel,matheus.alcantara}@sga.pucminas.br, cota@pucminas.br

Resumo. *O crescimento constante no volume de bases de dados em variadas áreas de pesquisa tem demandado arquiteturas de computadores mais eficazes para a utilização de algoritmos de mineração de dados, de maneira a realizar análises eficientes dessas bases. Mais desempenho é necessário, e arquiteturas mais poderosas tendem a consumir mais energia, acrescentando desafios para os projetos de hardware de processadores. Dessa forma, o projeto de novas arquiteturas com eficiência energética se faz necessário. Este trabalho propõe o projeto e avaliação de uma arquitetura em VHDL e FPGA para o algoritmo de clusterização K-means, visando alto desempenho em arquiteturas heterogêneas. Os resultados mostram que a implementação proposta apresenta uma redução de 91% em relação número de ciclos executados por um processador Intel Xeon E5-2620, consumindo até 95% menos energia.*

1. Introdução

O uso de técnicas para o tratamento de dados produzidos por diversas áreas, tais como processamento de imagens, bioinformática, previsão do tempo e redes sociais, tem produzido resultados com informações complexas, volumosas e heterogêneas. O conceito de *Big Data* surgiu, com o objetivo de tratar esses dados, que possuem estruturas variadas, além de serem obtidos a todo momento. O tratamento desses dados deve ser realizado em tempo hábil, para, por exemplo, gerar resultados para a tomada de decisões. Os processos de descoberta de dados em *Big Data* buscam encontrar padrões e similaridades entre os dados, a partir de análises computacionais, de modo a organizá-los de uma maneira lógica de acordo com suas características.

Realizar o processo de tratamento em tempo hábil demanda alto desempenho computacional, motivo pelo qual os algoritmos pertinentes à área são executados em poderosos processadores, como Unidades de Processamento Gráfico, do inglês *Graphics Processing Units* (GPUs) e o Intel Xeon Phi [Lee et al. 2016], além de arquiteturas heterogêneas, e.g., processadores convencionais + *Field Programmable Gate Array* (FPGA) [Neshatpour et al. 2016], bastante utilizados em computação de alto desempenho. Uma outra abordagem é a utilização de *clusters* de computadores, algumas vezes produzidos de maneira não convencional, e.g., um *cluster* de placas *Raspberry Pi*, buscando redução de custos e economia de energia. [Saffran et al. 2017].

O *K-means* é um algoritmo de clusterização muito utilizado para problemas de *Big Data*, devido sua eficiência. Porém, o tratamento de um grande volume de dados com

alta dimensionalidade não é uma tarefa trivial. Os algoritmos utilizados no contexto de *Big Data* geralmente necessitam de alto desempenho para atingirem seus objetivos em tempo hábil. Dessa forma, FPGAs tem sido alvo da academia e indústria, como alternativa para atender à demanda por desempenho e por soluções em tempo real [Dollas 2014]. Desenvolvedores e arquitetos esbarram em algumas limitações de *hardware*, como tipo, quantidade de bits e número de dimensões dos dados suportados, quando realizam a implementação do algoritmo *K-means* e outros relacionados a mineração de dados em dispositivos não convencionais.

Desta forma, o objetivo deste trabalho é o projeto da arquitetura do algoritmo *K-means* em um *hardware* programável, usando a linguagem de descrição *VHSIC Hardware Description Language* (VHDL). As contribuições dessa pesquisa compreendem uma arquitetura com: (i) suporte a dados de entrada de 64 bits; (ii) suporte a ponto fixo ou ponto flutuante; e (iii) flexibilidade na alteração dos principais parâmetros do algoritmo (número de pontos, centroides, dimensões e iterações) em tempo de operação do *hardware*, sem a necessidade de uma nova compilação para uma nova carga de execuções. Além disso, é apresentado um comparativo do *hardware* implementado com o algoritmo em *software*, em termos de eficiência energética, e uma análise de escalabilidade da solução.

Este artigo está organizado da seguinte maneira. A Seção 2 apresenta uma revisão da literatura, apresentando também alguns trabalhos correlatos. A Seção 3 demonstra a arquitetura proposta e o funcionamento do *hardware* desenvolvido. Na Seção 4 a metodologia adotada no desenvolvimento do projeto é apresentada. A Seção 5 contém os resultados encontrados e, na Seção 6 são apresentadas as considerações finais do trabalho.

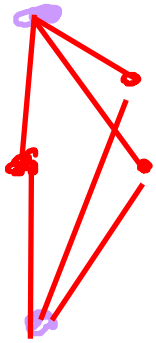
2. *Background*

Na busca por alto desempenho para aplicações de *Big Data* há o emprego de arquiteturas não convencionais. Como exemplo, um *cluster* de *Raspberry Pi* para algoritmos de mineração de dados pode ser usado como alternativa para alto desempenho com eficiência energética, quando comparada com a utilização de arqutieturas mais robustas como o Intel Xeon Phi [Saffran et al. 2017].

Da mesma maneira, FPGAs podem ser utilizados para solucionar os mesmos problemas, dada a sua flexibilidade e desempenho [Dollas 2014]. Um FPGA é um circuito integrado que permite ao desenvolvedor reconfigurar o *hardware* após sua fabricação, modificando o funcionamento do circuito. Para reconfigurar, ou reprogramar o FPGA, utilizam-se linguagens de descrição de *hardware* (HDL). O código HDL é compilado, para que a o *hardware* descrito seja implementado em um FPGA.

O uso de linguagens de descrição em *hardware* reconfigurável permite que algoritmos passíveis de implementação em linguagens de programação também possam ser implementados em *hardware*. Embora existam alguns desafios, essa característica é uma vantagem, dado que o processamento em *hardware* tende a ser mais rápido do que em *software*. Partindo desse princípio, o algoritmo *K-means* também pode ser desenvolvido para um FPGA, que é a proposta do presente artigo.

O *K-means* é um algoritmo usado em mineração de dados para particionamento de dados em grupos, ou *clusters*, de acordo com a similaridade entre eles. O algoritmo



trabalha com n pontos com d dimensões espaciais, que são agrupados em k *clusters*, levando-se em consideração a menor distância entre os pontos e os centros de cada *cluster*, chamados de centroides [Lloyd 1982].

Dado o conjunto de n pontos (objetos) e as d dimensões (atributos), define-se a quantidade k de *clusters* que serão gerados. Em seguida, é feita a inicialização dos centroides, a partir de pontos estratégicos (normalmente aleatórios). Na etapa seguinte é calculada a distância euclidiana entre cada ponto com cada centroide, vinculando o ponto ao centroide mais próximo. Por fim, os valores de cada centroide são atualizados, calculando-se a média de todos os pontos mapeados para o centroide verificado. Os dois últimos passos se repetem até que os valores dos centroides se mantenham constantes ou até que se atinja um número pré-determinado de iterações. Após a finalização das iterações, o mapeamento final dos dados em cada *cluster* é produzido. A Figura 1 exibe um exemplo obtido com a execução do algoritmo com $k = 4$.

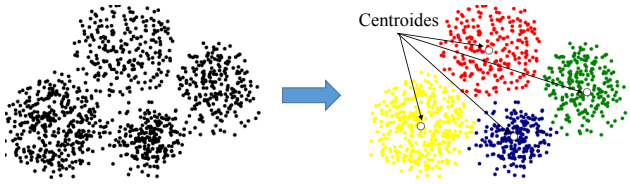


Figura 1. Resultado do algoritmo *K-means* com $k = 4$

A seção seguinte apresenta alguns trabalhos correlatos à este, que realizaram avaliações do algoritmo *K-means* em dispositivos que visam alto desempenho.

2.1. Trabalhos correlatos

No trabalho desenvolvido em [Baydoun et al. 2016] são apresentadas implementações do *K-means* com a biblioteca *OpenMP* utilizando CPU, e também com a biblioteca *CUDA* utilizando GPU. A avaliação dos autores utilizou entradas e parâmetros distintos, demonstrando bons resultados ao sugerir otimizações nas implementações do algoritmo, com a implementação em CUDA sendo a mais eficiente ao se aumentar o número de *clusters*.

De forma similar, em [Lee et al. 2016] o algoritmo *K-means* em paralelo foi avaliado no processador Intel Xeon Phi. Os autores exploraram as técnicas específicas para este processador para obter mais desempenho. As técnicas baseiam-se na organização dos conjuntos de dados em memória de maneira estratégica, para otimizar o processamento paralelo em nível de dados e *threads*.

Uma arquitetura do algoritmo *K-means* para FPGA é proposta em [Kutty et al. 2013]. O objetivo principal é a operação em alta velocidade com frequências de até 400 MHz. Para atingir o objetivo, a arquitetura armazena os dados de entrada em uma *Block RAM* inserida no FPGA; utiliza a distância de Manhattan para o cálculo das distâncias entre os pontos e os centroides; e os dados são de 8 bits. Contudo, a implementação limita o número de *clusters* do algoritmo entre 7 e 9.

Os autores em [Lin et al. 2012] também propõem a implementação em FPGA, visando dados com alta dimensionalidade. São aplicados conceitos de desigualdade triangular, além da eliminação do cálculo de raiz quadrada da distância euclidiana. A partir de dados com 8 bits, 6 *Lookup Tables* (LUTs) e uma memória DDR3 de 512MB externa,

foi possível processar dados de até 1024 dimensões, com desempenho até 17,5% melhor que o *benchmark* MNIST.

Os trabalhos citados que utilizaram FPGA, embora apresentem boas estratégias para a implementação do algoritmo *K-means*, deixam brechas para pesquisas, que são endereçadas pelo presente trabalho. Como exemplo, não foram encontrados experimentos que utilizem o *K-means* para FPGA com dados de 64 bits, sendo de ponto flutuante ou ponto fixo, e flexíveis em termos de alteração dos principais parâmetros (o número de *clusters*, pontos, dimensões e iterações) em tempo de operação. Além disso, nenhum dos trabalhos correlatos encontrados realizou uma análise da eficiência energética da solução.

3. Projeto da arquitetura do algoritmo *K-means*

A arquitetura do *hardware K-means* foi projetada e descrita em VHDL, com foco em FPGA. Uma visão geral da arquitetura é mostrada na Figura 2.

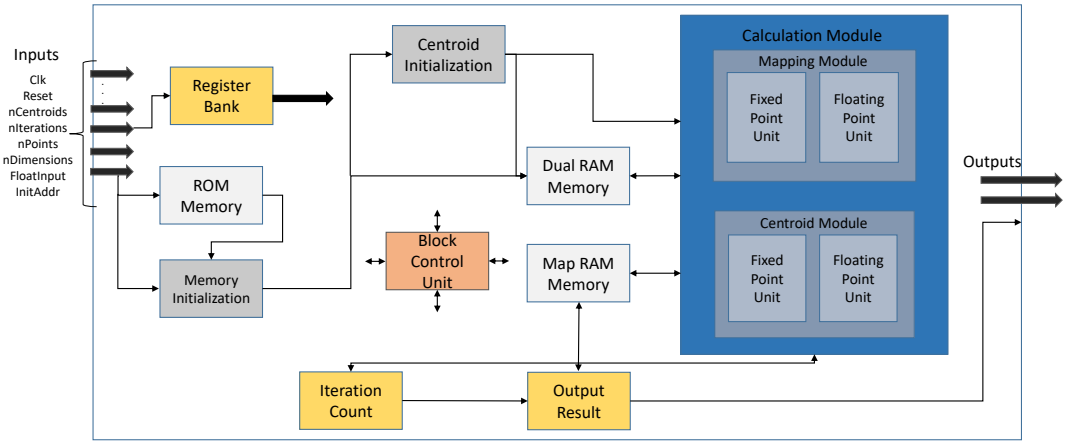


Figura 2. Arquitetura do *hardware K-means*

A arquitetura é baseada em blocos interligados e com funções específicas, gerenciados por uma unidade de controle central. Os blocos de operação desta arquitetura seguem um fluxo de execução determinado pelo diagrama de estados da Figura 3, sendo que a cada pulso de *clock*, o controlador determina qual será o próximo bloco ativo.

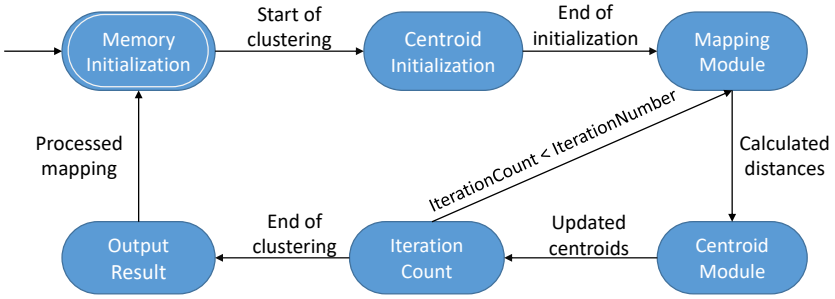
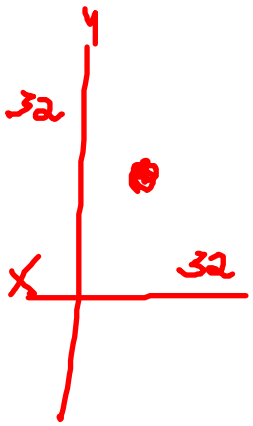


Figura 3. Diagrama de estados do *hardware*

A arquitetura depende de algumas entradas, como *clock* e *reset*; número de centroides, pontos, iterações e dimensões; se os dados serão de ponto fixo ou flutuante (Flo-



atInput); e o endereço na memória ROM do primeiro ponto (InitAddr). As suas saídas (Saida e EnSaida) representam o mapeamento dos pontos em cada grupo.

Para este protótipo, a base de dados é fornecida pelo usuário e armazenada na memória ROM, em tempo de compilação. Os dados dessa base devem ser de 64 bits, sendo que cada 32 bits representa o valor de uma dimensão de cada ponto. Ou seja, caso um ponto tenha 2 dimensões, ele será representado por 1 dado, caso tenha 4 dimensões por 2 dados. Para pontos do tipo ponto fixo, cada dimensão tem o seu valor correspondente em memória. Já para pontos do tipo ponto flutuante, cada dimensão deve ser codificada no padrão IEEE 754 (precisão simples) [Hennessy and Patterson 2014].

3.1. Módulo de Controle

Para controlar o fluxo de operação dos componentes garantindo a correta execução do algoritmo, foi elaborado um bloco de controle, com várias sub-unidades. Cada componente da arquitetura possui uma entrada de habilitação gerenciada pela unidade de controle do *hardware*. As unidades do bloco de controle são descritas a seguir:

Block Control Unit: Unidade de controle global de operação de todos os blocos da arquitetura, que gerencia, habilita e define, a cada pulso de *clock*, qual bloco irá executar sua operação. Recebe como entrada o valor correspondente ao próximo bloco a ser habilitado, e informa a todos os componentes quais as ações que devem realizar.

Iteration Count: Responsável por verificar a quantidade de iterações já realizadas no algoritmo, determinando a sua condição de parada. Possui um contador que compara seu valor com a quantidade definida na entrada, e é incrementado a cada iteração, até o final do processamento, quando o resultado é enviado para a saída.

Output Result: Este componente é acionado após a execução de todas as iterações do *K-means*. Possui um contador que é incrementado a cada pulso de *clock* até atingir o número total de pontos, lendo os dados da *Map RAM Memory*, e os enviando para a saída. Durante sua execução, o pino EnSaida é habilitado informando que o mapeamento final de cada um dos pontos está sendo exibido.

Register Bank: Responsável por receber os dados de entrada (número de centroides, iterações, pontos e dimensões) e armazená-los em registradores internos, que estão disponíveis para as operações que necessitarem. Também atribui a um registrador o endereço de memória inicial para o armazenamento e leitura dos centroides na *Dual RAM Memory*, demarcando a divisão entre pontos e centroides armazenados de forma contígua.

3.2. Módulo de Memórias

Memórias embutidas no *design* da arquitetura, que permitem o acesso mais rápido aos dados, em comparação com a utilização de memórias externas. Assim, utilizando a ferramenta *IP Catalog* do *software* Quartus da Intel/Altera, projetou-se uma memória ROM e duas memórias RAMs.

ROM Memory: Memória ROM que armazenará a base de dados a ser avaliada pelo algoritmo, permitindo o armazenamento de até 16384 dados de 64 bits. Esta memória é utilizada somente para facilitar as avaliações do *hardware* no FPGA de prototipação, pois o objetivo é utilizar memórias externas para armazenar a base de dados.

Dual RAM Memory: Memória RAM com capacidade de até 16384 dados de 64 bits, utilizada para armazenar os pontos da base de dados que serão processados e os valores dos centroides que serão calculados. Esta memória possui dois canais de entrada e dois de saída (A e B), que permitem a leitura e escrita de duas informações em um mesmo ciclo de *clock*, desde que não sejam escritas em um mesmo endereço. Além disso, é organizada de modo a armazenar dinamicamente os pontos, seguidos dos centroides.

Map RAM Memory: Memória RAM utilizada para armazenar o mapeamento final dos pontos de entrada. Possui capacidade de até 16384 dados de 10 bits, podendo receber valores entre 0 e 1023, correspondentes aos centroides mapeados.

3.3. Módulo de Inicialização

No algoritmo *K-means*, é necessário definir inicialmente os centroides de cada *cluster*. Pensando na necessidade de alto desempenho do *hardware*, foi elaborado um bloco de inicialização que realiza a preparação para a execução das etapas do algoritmo. Este bloco é subdivido da seguinte forma:

Memory Initialization: Etapa inicial, na qual, por meio de um contador, o endereço de um ponto a ser lido é enviado para a memória ROM, a partir de um endereço inicial (InitAddr). Em seguida, a memória ROM lê o dado solicitado e o envia para a *Dual RAM Memory*, que armazena o valor recebido em seu primeiro endereço. A cada pulso de *clock*, o contador é incrementado para buscar um novo dado na ROM, que será salvo no próximo endereço da RAM. Este processo ocorre até que todos os pontos sejam armazenados na *Dual RAM Memory*.

Centroid Initialization: No algoritmo *K-means* convencional, a escolha dos pontos que serão centroides, inicialmente, é feita aleatoriamente. Porém, a estratégia utilizada neste projeto consiste em atribuir para os centroides os valores dos k pontos iniciais da base dados, reduzindo a complexidade e latência do *hardware*, sem prejuízos no resultado. Este componente possui um funcionamento baseado em estados, com um contador de centroides, um controlador e um bloco de inicialização. O controlador envia o endereço de um ponto para a *Dual RAM Memory*, que, em seguida, envia o valor lido para o bloco de inicialização. O bloco define o endereço da *Dual RAM Memory* em que o centroide será armazenado. O contador é incrementado e o centroide é salvo na memória. O processo é repetido até que todos os k centroides sejam iniciados.

3.4. Módulo de Cálculo do Mapeamento

A Figura 4 mostra a etapa principal do algoritmo *K-means*, que engloba o cálculo da distância entre pontos e centroides. Este bloco busca encontrar, para cada ponto, o centroide mais próximo dele, atualizando este mapeamento na *Map RAM Memory*. Este processo é repetido até que todos os pontos tenham sido mapeados.

O *hardware* projetado processa em paralelo duas dimensões de cada ponto de entrada, acelerando o cálculo das distâncias. O módulo de cálculo do mapeamento possui duas unidades de processamento com as mesmas funcionalidades e lógica de execução, sendo que uma unidade é responsável por operações com dados do tipo ponto fixo e outra que trabalha com dados do tipo ponto flutuante.

A distância euclidiana foi escolhida, pois produz uma melhor acurácia no resultado para o algoritmo *K-means* [Estlick et al. 2001], porém, uma adaptação foi realizada.

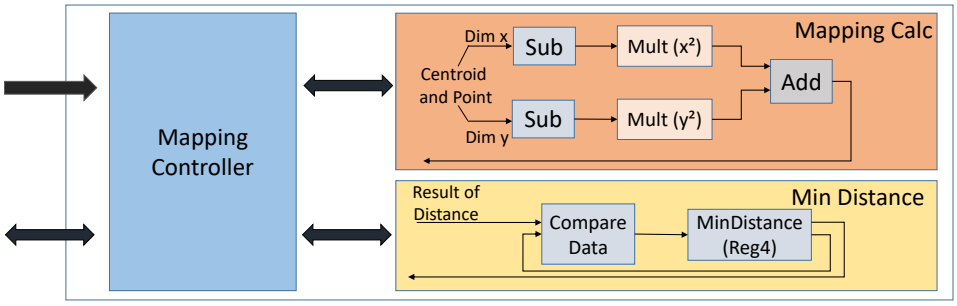


Figura 4. Módulo de Cálculo do Mapeamento

Essa distância é dada pela fórmula $d(p,c) = \sqrt{\sum_{k=1}^{n} |p_k - c_k|^2}$, em que n representa o número de dimensões e p_k e c_k representam o k-ésimo atributo do ponto p e do centroide c que são comparados. Contudo, a implementação do cálculo de raiz quadrada adiciona complexidade extra ao *hardware*, tal como identificado em [Lin et al. 2012].

Como é necessário encontrar somente a menor distância entre um ponto e um centroide, foi removido o cálculo da raiz quadrada, produzindo uma solução aproximada que não prejudica o mapeamento final. Foi usado então o valor da distância sem a raiz quadrada, obtido pela equação adaptada: $d(p,c)^2 = \sum_{k=1}^{n} |p_k - c_k|^2$.

Cada bloco de cálculo de distância tem dois multiplicadores e um somador que verificam duas dimensões em paralelo, armazenando a soma dos quadrados em um acumulador (*distanceCalc*). Enquanto houver dimensões para serem verificadas, o *hardware* calcula a distância destas próximas dimensões e soma seus resultados com os valores armazenados no acumulador. Ao final do cálculo das distâncias, o valor armazenado no *distanceCalc* é comparado com a distância mínima quadrática encontrada até o momento, que está armazenada em um registrador (Reg4). Se a nova distância for menor do que o valor de Reg4, este é atualizado e o centroide correspondente é mapeado para o ponto, de modo que seu valor é armazenado em outro registrador (Reg7). Este processo é repetido até que todos os centroides tenham sido comparados com o ponto selecionado. Ao terminar estas operações, o valor armazenado em Reg7 é mapeado para o ponto e gravado no endereço correspondente na *Map RAM Memory*. Paralelamente, o próximo ponto é carregado no bloco de cálculo de distâncias e um novo ciclo de verificações é executado, repetindo até que todos os pontos tenham sido mapeados nos centroides.

3.5. Módulo de Cálculo dos Centroides

O módulo de cálculo de centroides, exibido na Figura 5, possui três vetores para armazenar a soma dos atributos. O processamento ocorre de acordo com uma máquina de estados que coordena as unidades de soma e de cálculo de média. O módulo entra em operação após o término do cálculo das distâncias, sendo responsável por processar e atualizar os valores dos centroides, armazenados na região inferior da *Dual RAM Memory*.

Este módulo é constituído de dois componentes, um para dados de ponto fixo e outro para dados de ponto-flutuante, da mesma forma que no módulo de cálculo do mapeamento. A seguir, descreve-se a sua máquina de estados, em sua ordem de execução.

IDLE: Estado de espera do módulo, até receber um sinal para iniciar.

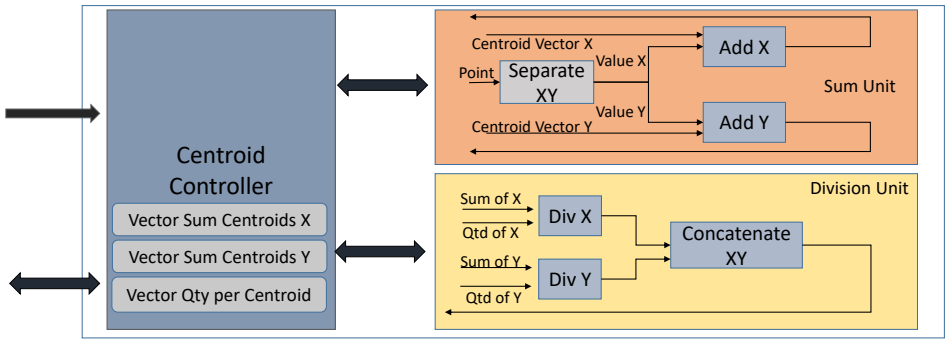


Figura 5. Módulo de Cálculo dos Centroides

LOADDATA: Neste, é verificado se todos os dados já foram lidos. Se sim, o estado VERIFYDIV é acionado. Se não, é solicitada a leitura das dimensões dos pontos na *Dual RAM Memory* e do mapeamento na *Map RAM Memory*.

WAITDATA: Aguarda até que a leitura dos dados solicitados seja realizada.

SUM: Aciona a unidade de soma, em que os valores de duas dimensões são somados e armazenados em vetores. Cada posição do vetor corresponde a um centroide. O processo é executado até todos os pontos serem verificados, em seguida vai para o estado LOADDATA.

VERIFYDIV: Neste estado, é verificado se todos os centroides já foram validados. Se sim, aciona-se o estado VERIFYDIMENSION. Se não, o estado DIV.

DIV: Responsável por controlar a unidade que calcula a média dos pontos mapeados para cada centroide. Esta unidade possui dois componentes de divisão, projetados na ferramenta *IP Catalog*, que calculam em paralelo, a média de atributos de duas dimensões, que ao final são concatenados em um dado de 64 bits. Caso os valores sejam de ponto flutuante, as médias são calculadas mantendo-se os valores decimais. Se forem do tipo ponto fixo, os valores são arredondados, considerando apenas a parte inteira do resultado da divisão.

STOREDATA: Neste estado, os valores dos centroides são atualizados com os novos resultados, em seus endereços correspondentes na *Dual RAM Memory* e em seguida retorna para o estado VERIFYDIV.

VERIFYDIMENSION: Estado em que é verificado se todas as dimensões dos pontos foram calculadas. Se sim, aciona o estado ENDSTATE, caso contrário, as próximas dimensões são lidas, retornando ao estado LOADDATA.

ENDSTATE: Encerramento do módulo, após atualização de todos os centroides.

4. Metodologia de avaliação

A partir da apresentação da arquitetura proposta do algoritmo *K-means*, a descrição do *hardware* respectivo foi realizada em VHDL. A sintetização e compilação foram feitas no *software* Quartus Prime Lite 16.1. O *hardware* foi implementado no FPGA *Intel/Altera Cyclone IV-E EP4CE115F29C7*, contido na placa DE2-115. A estratégia para avaliação do *hardware* compreendeu 4 etapas.

Na 1ª etapa foram geradas bases de dados sintéticas, com valores pseudoaleatórios de uma distribuição normal para cada dimensão, entre 0 e 65536.

A 2ª etapa compreendeu a comparação do número de ciclos de *clock* gastos na execução do *K-means* com dados de ponto fixo para FPGA e com dados de ponto flutuante para FPGA e para o *software* do *CAP Bench* [Souza et al. 2017] executado com 12 *threads*, com diferentes tamanhos da base de dados de entrada. A quantidade de centroides (k) variou entre 2, 4 e 8. Já a quantidade de pontos, variou entre 256, 1024 e 4096. Nesta etapa, o número de dimensões de cada ponto (seus atributos) foi mantido em 4, da mesma forma que o número de iterações do algoritmo (também 4). Utilizou-se um computador com 2 processadores Intel Xeon E5-2620 de 2.10GHz, com 6 núcleos cada, 32 GB de memória RAM, Linux CentOS, e GCC versão 4.9.2.

Na 3ª etapa, fixou-se o número de pontos, iterações e centroides, variando-se a quantidade de dimensões, comparando os ciclos gastos pelo FPGA e pelo *software*.

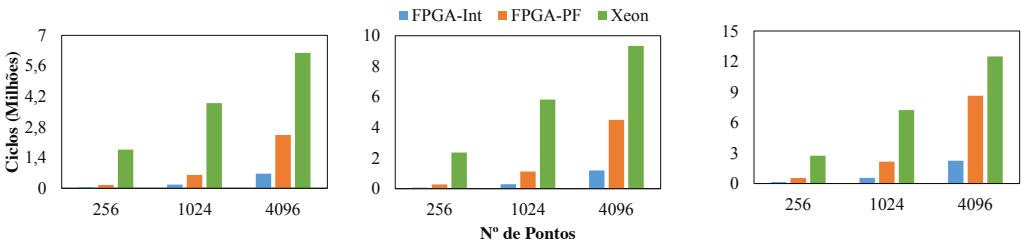
A comparação do tempo de execução e do consumo de energia das duas implementações foi feita na 4ª etapa, com as mesmas bases de dados da 2ª etapa. Também fixou-se a base de dados, porém variando a frequência de processamento do FPGA. Foram utilizadas as ferramentas *Intel PowerPlay EPE* e PAPI versão 5.5.0 para verificar o consumo.

5. Resultados

Nesta seção, são discutidos os resultados obtidos no trabalho. A síntese do *hardware* apresenta os valores mostrados na Tabela 1, referentes a total de multiplicadores, registradores, elementos lógicos e bits de memória disponíveis e utilizados pelo *hardware* no FPGA usado.

Tabela 1. Elementos do *Hardware*

Elementos	Qtde. do FPGA	Qtde. Arq. *K-means*	Uso FPGA (Arq. *K-means*)
Multiplicadores	532	62	12%
Registradores	114480	6866	6%
Lógicos	114480	15000	13%
Bits Mem.	3981312	2270602	57%



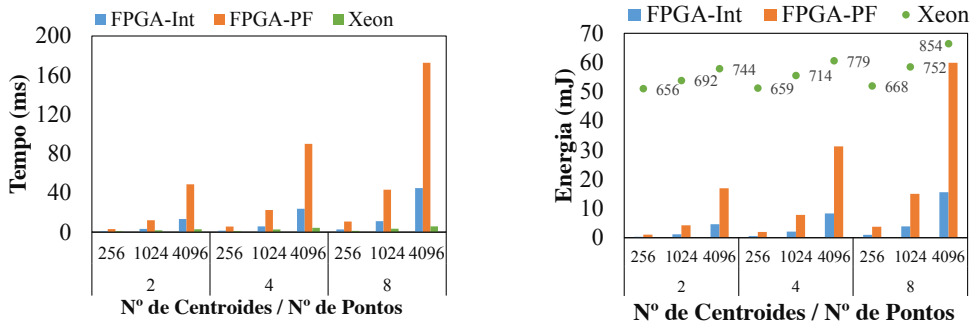
(a) 2 centroides (b) 4 centroides (c) 8 centroides

Figura 6. Quantidade de ciclos por Número de Pontos

O desempenho do *hardware* comparado com a implementação em *software*, em número de ciclos de *clock*, é apresentado na Figura 6, que mostra comportamentos semelhantes ao variar a carga de dados em cada um dos gráficos apresentados. O objetivo

é verificar o desempenho ao utilizar dados de ponto flutuante (FPGA-PF e Xeon) e de ponto fixo (FPGA-Int). Assim, o FPGA consegue um desempenho melhor em relação ao Xeon ao ter uma redução, no pior caso, de 82% e 31%, e no melhor caso de 98% e 91% no número de ciclos, respectivamente com dados de ponto fixo e flutuante. Após a clusterização é obtido um resultado final com 98% de semelhança na distribuição dos dados ao comparar o FPGA-PF com o Xeon, e 70% de semelhança ao comparar FPGA-Int com FPGA-PF, mostrando que mesmo ao se utilizar truncamento são obtidos resultados satisfatórios que justificam a utilização de dados de ponto fixo, já que os cálculos de ponto flutuante são mais complexos. Constata-se também que o *hardware* suporta o aumento da carga de trabalho adequadamente.

De modo a verificar o desempenho da arquitetura ao se variar o número de dimensões dos dados de entrada em 2, 4, 6 ou 8, foram realizadas execuções no FPGA e no Xeon, fixando o número de pontos em 1024, o de centroides e o de iterações em 4. Com dados de ponto fixo gasta-se 6% e com dados de ponto flutuante 20% do número de ciclos executados pelo processador, mostrando que a quantidade de dimensões não interfere na melhoria adquirida com o uso do FPGA.



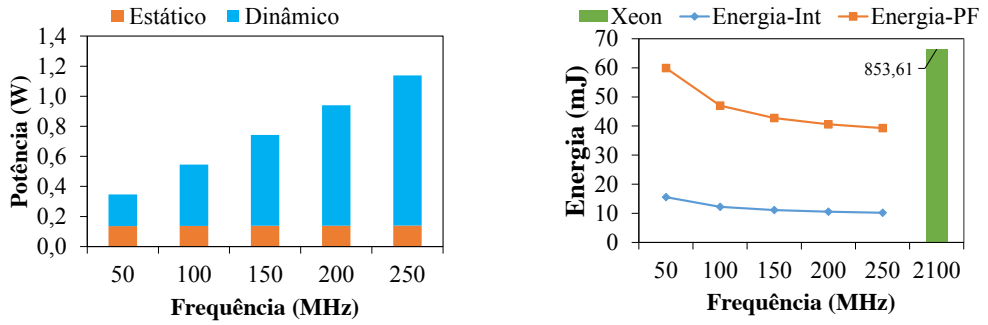
(a) Tempo de execução

(b) Energia consumida

Figura 7. Tempo e energia com variação da carga de dados

Foram comparados os tempos de execução do *K-means* em FPGA (50MHz) e no processador Xeon (2100MHz), com o intuito de mostrar, que embora a arquitetura proposta execute o algoritmo em uma quantidade de ciclos menor, a frequência do FPGA utilizado pode interferir consideravelmente no tempo de resposta da aplicação, como pode ser verificado na Figura 7(a), onde o tempo de execução do Xeon é até 80% menor que o do FPGA-Int e até 90% menor que o do FPGA-PF. Além dessa avaliação, foi medido o consumo de energia, em milijoules, em dois tipos de análises. O primeiro tipo baseou-se em se variar a carga de dados de entrada, utilizando o FPGA com frequência de 50MHz, mostrando que o FPGA consome em seu pior caso, 2% com dados de ponto fixo e 8% com ponto flutuante, e em seu melhor caso 1% em ambos os tipos de dados, da energia consumida pelo Xeon, mostrado na Figura 7(b).

O outro tipo mostra uma projeção do consumo de energia do FPGA ao variar sua frequência de operação mantendo uma carga fixa de dados de ponto flutuante com 4096 pontos, 8 centroides, 4 dimensões e 4 iterações. O FPGA possui um gasto de potência estático, *hardware* ligado sem executar operações, e outro dinâmico, *hardware* ligado processando operações, que cresce à medida que frequência aumenta, mostrado na Fi-



(a) Potência por frequência (FPGA-PF)

(b) Energia por frequência

Figura 8. Consumo por frequência

gura 8(a). O resultado de energia consumida do Xeon e do FPGA com ponto fixo e flutuante, é comparado na Figura 8(b), mostrando que o FPGA tende a consumir menos energia à medida que a frequência aumenta, pois mesmo com o aumento de potência dinâmica, o tempo de processamento reduz, diminuindo consequentemente a energia consumida. Além disso, é possível verificar que o FPGA consome em seu melhor e pior caso, respectivamente 95% e 93% com dados de ponto flutuante, e 98% com dados de ponto fixo, menos energia do que o processador.

6. Conclusão e trabalhos futuros

Neste artigo é apresentado o projeto de uma arquitetura de *hardware* do algoritmo *K-means* e realizada sua implementação em FPGA. O projeto desenvolvido possui uma entrada de dados de 64 bits, com atributos de 32 bits, podendo ser de ponto fixo ou flutuante, que permitem que o usuário altere o número de pontos, *clusters*, iterações e dimensões em tempo de operação do *hardware*. As estratégias do processamento de dois atributos em paralelo e no cálculo de distância euclidiana tornam o *hardware* proposto mais robusto.

A avaliação de desempenho do projeto em comparação ao *benchmark* CAP Bench, executado em uma máquina com processador Xeon, mostra que o FPGA apresenta um ganho em relação ao Xeon, no melhor caso, de 98% e 91% do número de ciclos, respectivamente com dados de ponto fixo e flutuante. Ao avaliar o tempo de execução do algoritmo em um FPGA com 50MHz e no Xeon com 2100MHz, nota-se que a frequência do dispositivo utilizado interfere diretamente no tempo de resposta da aplicação, onde o Xeon apresenta um tempo de até 90% menor, em comparação com o FPGA-PF. Já em relação a avaliação de consumo de energia, o FPGA-PF consome, em seu melhor caso, 95% menos energia do que o processador. A contribuição deste trabalho é a elaboração de uma arquitetura de *hardware* e sua implementação em VHDL para FPGA, do algoritmo *K-means*, com suporte a dados de análise com valores de ponto fixo ou ponto flutuante de 64 bits, não encontrado em trabalhos correlatos, demonstrando sua avaliação de desempenho e energia.

Os resultados apresentados são promissores e para trabalhos futuros o protótipo será avaliado em outros FPGAs, e.g. Arria 10. O projeto visa um funcionamento em frequências mais altas com variação da carga de trabalho, baixo tempo de execução, es-

calabilidade e eficiência em desempenho e energia. Além disso, há também a retirada da memória ROM da arquitetura, a melhoria das etapas de cálculo de mapeamento e cálculo de centroides para execução em paralelo de um número superior a 2 atributos, substituição do número de iterações como ponto de parada da clusterização na arquitetura, pela comparação da distância mínima de modificação dos valores dos centroides e verificação da possibilidade de se utilizar a distância de Manhattan no cálculo de mapeamento.

Agradecimentos

Os autores agradecem ao CNPq, FAPEMIG, CAPES e a PUC Minas pelo suporte no desenvolvimento do trabalho.

Referências

Baydoun, M., Dawi, M., and Ghaziri, H. (2016). Enhanced parallel implementation of the k-means clustering algorithm. In *3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*, pages 7–11.

Dollas, A. (2014). Big data processing with fpga supercomputers: Opportunities and challenges. In *IEEE Computer Society Annual Symposium on VLSI*, pages 474–479.

Estlick, M. et al. (2001). Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *9th International Symposium on FPGA*, pages 103–110. ACM.

Hennessy, J. L. and Patterson, D. A. (2014). *Organização e Projeto de Computadores: a interface hardware/software*, volume 4. Elsevier Brasil.

Kutty, J. S. S., Boussaid, F., and Amira, A. (2013). A high speed configurable fpga architecture for k-mean clustering. In *IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 1801–1804.

Lee, S. et al. (2016). Evaluation of k-means data clustering algorithm on intel xeon phi. In *IEEE International Conference on Big Data (Big Data)*, pages 2251–2260.

Lin, Z., Lo, C., and Chow, P. (2012). K-means implementation on fpga for high-dimensional data using triangle inequality. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 437–442.

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

Neshatpour, K., Sasan, A., and Homayoun, H. (2016). Big data analytics on heterogeneous accelerator architectures. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–3.

Saffran, J. et al. (2017). A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms. In *Desprez F. et al. Euro-Par 2016: Parallel Processing Workshops. Euro-Par 2016*, Lecture Notes in Comp. Science, vol 10104, Springer, Cham.

Souza, M. A. et al. (2017). Cap bench: a benchmark suite for performance and energy evaluation of low-power many-core processors. *Concurrency and Computat.: Pract. Exper.*, 29:e3892. doi: 10.1002/cpe.3892.