A GPU/FPGA-based K-means clustering using a parameterized code generator

Jeronimo Penha
Department of Informatics
CEFET-MG
Leopoldina, MG, Brazil
jpenha@cefetmg.br

Lucas Bragança
Department of Informatics
UFV
Florestal, MG, Brazil
lucas.braganca@ufv.br

Kristtopher Coelho
Department of Informatics
UFV
Florestal, MG, Brazil
kristtopher.coelho@ufv.br

Michael Canesche
Department of Informatics
UFV
Viçosa, MG, Brazil
michael.canesche@ufv.br

Jansen Silva
Department of Informatics
UFV
Viçosa, MG, Brazil
jansen.silva@ufv.br

Giovanni Comarela
Department of Informatics
UFV
Viçosa, MG, Brazil
gcom@ufv.br

José Augusto M. Nacif
Department of Informatics
UFV
Florestal, MG, Brazil
jnacif@ufv.br

Ricardo Ferreira
Department of Informatics
UFV
Viçosa, MG, Brazil
ricardo@ufv.br

***Abstract*—The *K-means* algorithm is a method used for the unsupervised learning task of data clustering. This work presents a *K-means* specific domain code generator capable of generating code for GPUs and FPGAs. To increase efficiency, the code is parameterized and specialized for Nvidia GPUs and Intel/Altera CPU-FPGA HARP v.2 platform. Furthermore, the generator is modular and can be extended to other FPGA and GPU platforms. Another contribution of this work is to simplify the use of high performance FPGAs for programmers, once our generator does not require hardware knowledge in order to provide a high performance accelerator at the software level. The generator also simplifies GPU programming. In comparison to an Intel XEON CPU, our experiments show a 55x speed-up for the GPU execution time and a 13.8x speed up for the FPGA. With regard to energy, the FPGA was up to 10 times more efficient than the evaluated GPUs (Nvidia K40 and 1080ti).**

***Keywords*-FPGA, GPU, Accelerators, K-Means**

I. Introdução

Atualmente, as GPUs (*Graphics Processing Unit*) e os FPGAs (*Field-Programmable Gate Array*) são duas tendências em aceleradores para desempenho com eficiência energética em comparação aos processadores de uso geral [1]. Neste contexto, a eficiência pode ser maior ao trabalhar com domínios específicos, conforme destacado por David Patterson e John Hennessy na aula do prêmio Turing de 2017 [2]. Apesar dos FPGAs oferecerem maior potencial de eficiência energética, o maior desafio ainda é a sua programação que exige conhecimentos específicos de hardware, mesmo com os ambientes de alto nível como OpenCL [3].

Uma demanda atual é a extração de conhecimento e identificação de grupos de grandes volumes de dados. Este artigo aborda a implementação do algoritmo *K-means* [4] de aprendizado não supervisionado nas plataformas de GPU e FPGA. O algoritmo *K-means* possui potencial para a paralelização de operações com reúso de dados, que é uma característica apropriada para as arquiteturas alvo. A primeira contribuição deste trabalho é a apresentação de um gerador de código parametrizável para GPU para a execução do *K-means*, que otimiza o reúso dos dados. A segunda contribuição é a geração automática e transparente de hardware e software para execução do algoritmo na nova plataforma CPU-FPGA HARP v.2 de alto desempenho da Intel.

A estrutura do artigo é descrita, a seguir. A Seção II apresenta brevemente o algoritmo *K-Means*, a Seção III descreve a plataforma HARP v.2 e a Seção IV apresenta a arquitetura desenvolvida para execução em FPGA. Na Seção V, o gerador para GPU é apresentado. Os experimentos e resultados são discutidos na Seção VI, os trabalhos relacionados são discutidos na Seção VII, e por fim, a Seção VIII apresenta as principais conclusões.

II. *K-means*

O *K-means* é um algoritmo de aprendizado não supervisionado capaz de particionar um conjunto de pontos em k grupos. O algoritmo é utilizado em larga escala em aplicações de mineração de dados [5]. A cada iteração, todos os pontos são avaliados. Várias iterações são executadas até que convirja. Este processo exige esforço computacional para grandes volumes de dados [6].

A Figura 1 apresenta um exemplo com pontos de duas dimensões, x e y, dos quais deseja-se classificar em dois grupos, i.e., $k = 2$. Cada grupo terá um centroide C_i e cada ponto é classificado no grupo do centroide mais próximo. A Figura 1(a) mostra o ponto p a ser classificado no grupo G_0 por estar mais perto do centroide C_0. A etapa de classificação é executada para todos os pontos. A próxima etapa é o cálculo da nova posição dos centroides. Essa nova posição é a que minimiza a média do quadrado das distâncias de todos os pontos do grupo. A Figura 1(b) mostra o reposicionamento dos centroides após a execução de uma iteração. Também é destacado o reposicionamento

do centroide C_0 e os pontos classificados nos grupos G_0 e G_1 na etapa anterior. O algoritmo repete as duas etapas (classificação e reposicionamento) até que convirja ou atinja um limite de iterações.

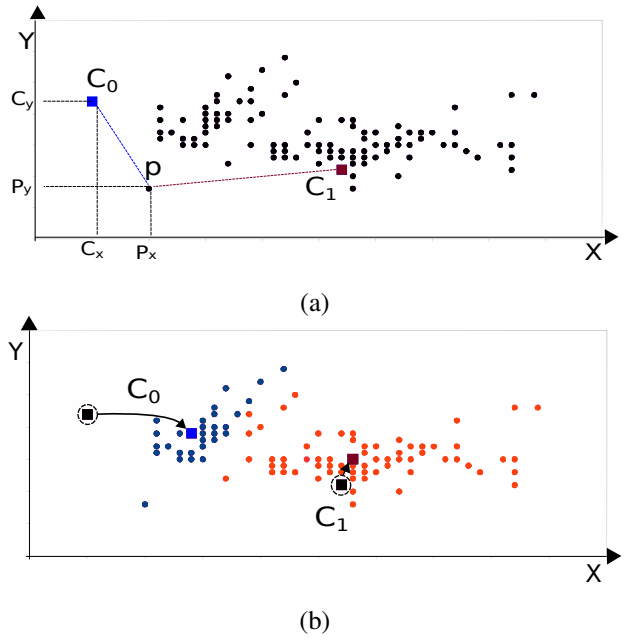

Figure 1. Exemplo de agrupamento pelo *K-means*.

O algoritmo pode ser descrito com o paradigma de *map-reduce*. A primeira etapa de classificação, pode ser definida por um mapeamento seguido de duas reduções. A Figura 2(a) mostra o fluxo de operações para o exemplo com duas dimensões ($n = 2$) e dois centroides ($k = 2$). O mapeamento é responsável pelo cálculo da distância em cada dimensão. A métrica de distância utilizada no *K-means* é a distância euclidiana, no qual, por simplicidade foi substituída pela sua norma. Assim, para a dimensão x no ponto P em relação ao centroide C_i, a distância será calculada como $D_{i_x} = (P_x - C_{i_x})^2$. Esta operação básica é executada para cada ponto e todas as suas dimensões em relação a todos os centroides. Como é uma métrica de distância, uma simplificação pode ser feita com a retirada do cálculo da raiz quadrada com o intuito de reduzir a complexidade [7]. O segundo passo é uma redução para somar as distâncias de todas as dimensões do ponto em relação ao centroide. No exemplo, para os centroides C_0 e C_1, o cálculo será $D_0 = D_{0_x} + D_{0_y}$ e $D_1 = D_{1_x} + D_{1_y}$, respectivamente. O terceiro passo é a redução para encontrar o centroide mais próximo de p, ou seja, $Min(D_0, D_1)$. A função Min retorna o número do centroide mais próximo de p. Esta é a fase de classificação do *K-means*. Formalmente, para k centroides e n dimensões, o ponto será classificado como $Min(D_0, \ldots, D_{n-1})$ onde a distância para o centroide C_i será $D_i = \sum_{j=0}^{n-1}(p_j - C_{i_j})^2$. Assim, tem-se kn subtrações e multiplicações durante o mapeamento, a redução de soma totaliza $k(n-1)$ adições e a redução de mínimo com $k-1$ comparações. A cada iteração para m pontos, tem-se $m[2kn + k(n-1) + k - 1]$ operações. A primeira fase do algoritmo é apropriada para execução em GPU e FPGA que são capazes de executar todas as operações em paralelo, dependendo do código, da arquitetura e dos valores de m, k e n. Outro aspecto favorável às GPUs e aos FPGAs é o reúso do dados que é definido pelo número de operações realizadas para cada dado lido da memória, que é igual $\frac{m[2kn+k(n-1)+k-1]}{mn} = 3k - \frac{1}{n}$, ou seja, quanto maior for o valor de k, o cenário se torna mais favorável às GPUs e aos FPGAs.

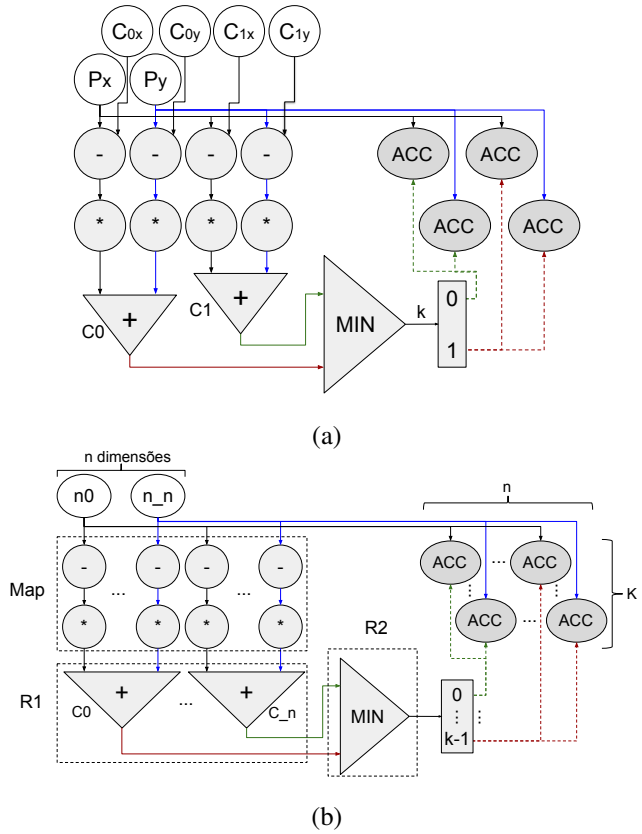

Figure 2. (a) Grafo de operações com $k = 2$ e $n = 2$; (b) Grafo Genérico.

A etapa de reposicionamento é responsável pelo cálculo dos novos valores para cada uma das dimensões dos centroides. Para cada ponto classificado pelo centroide C_i, os valores de cada dimensão são acumulados. Por isto há vários acumuladores (Acc) na Figura 2. Se t pontos forem classificados para o centroide C_i, a nova posição na dimensão j para o centroide C_i será calculado por $C_{i_j} = \frac{\sum_{j=1}^{t} p_j}{t}$.

III. Plataforma de Alto Desempenho com FPGA HARP v.2

Recentemente, várias plataformas para computação de alto desempenho com FPGAs foram apresentadas por grandes empresas como Intel-Altera [8], Amazon [9] e

Microsoft [10]. A plataforma utilizada neste trabalho é o HARP v.2 da Intel-Altera, que possui um FPGA fortemente acoplado através da memória compartilhada com um processador Xeon E5-2600 v4. A Intel provê interfaces em *software* e *hardware* que mantêm a coerência de dados entre o processador e o FPGA.

A Figura 3(a) ilustra os principais pontos da arquitetura da plataforma HARP v.2. A Intel provê a API OPAE (*Open Programmable Acceleration Engine*) que traz os objetos necessários para a comunicação do aplicativo com os aceleradores desenvolvidos para a plataforma. Entretanto, é necessário que o usuário possua conhecimentos específicos em FPGA e da documentação do OPAE. A ligação física entre memória e o acelerador é feita através do barramento Intel QPI (*Quick Path Interconnect*) mais dois barramentos PCI-e, nos quais são possíveis taxas de transmissão de dados na ordem de 16GB/s. O FPGA mantém uma *cache* interna de 64 Kbytes. A coerência com as memórias é feita de forma transparente. A CPU e o FPGA trabalham de forma assíncrona e independente.

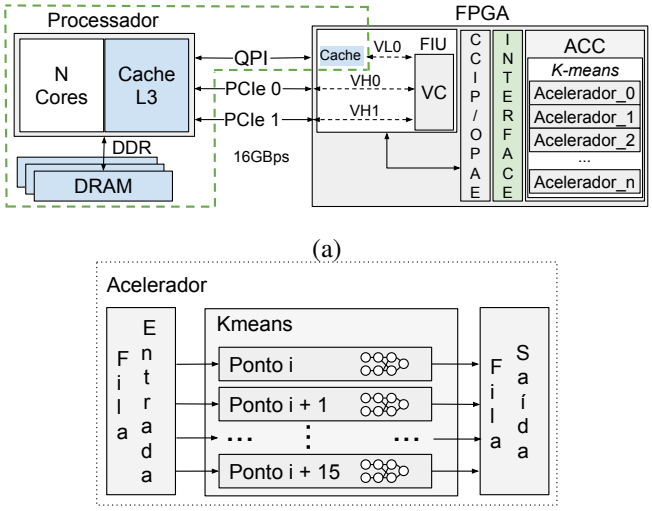

Figure 3. (a) Plataforma HARP v.2 da Intel (b) Arquitetura do acelerador.

Diferente da arquitetura de uma GPU que possui memória dedicada, o FPGA do HARP v.2 terá sempre que ler dados das memórias conectadas a CPU (*cache* L3 e RAM) através dos barramentos PCI e QPI, uma vez que a cache implementada dentro do FPGA é muito pequena, limitada a 64 Kbytes.

IV. Arquitetura do Acelerador *K-means* no FPGA

A arquitetura do gerador é o mapeamento direto do grafo de fluxo de dados do algoritmo de forma a explorar o potencial de paralelismo espacial e temporal (*pipeline*). Além do grafo, as interfaces de entrada e saída são adicionadas, ilustradas na Figura 3(b), que fazem o acoplamento da plataforma de hardware com o código em execução no processador implementado para plataforma HARP v.2 da Intel. A interface pode ser estendida para outras plataformas. As interfaces são baseadas na unidade de hardware apresentada em [11]. Como no HARP v.2, a granularidade da comunicação é limitada a uma linha de cache de 64 bytes, caso um ponto necessite menos de 64 Bytes, a arquitetura explora o paralelismo espacial com replicações, o que possibilita o processamento de vários pontos em paralelo para maximização da vazão de dados. Ademais, para mitigar a latência da comunicação, a interface possibilita o uso de filas e mais de uma instância do acelerador com execuções simultâneas.

Toda a arquitetura é gerada automaticamente de forma parametrizada em função do número de dimensões do conjunto de dados e do número de centroides. O gerador foi desenvolvido em Python com auxílio da biblioteca Veriloggen [12]. O hardware é gerado em Verilog e sintetizado com a ferramenta Intel Quartus para FPGAs. O gerador encapsula toda a complexidade de interface, sincronismo, compilação e síntese para FPGAs e a comunicação do hardware/software. O usuário precisa apenas instanciar os dados a serem processados e fazer a chamada ao acelerador a partir de software.

V. Gerador de código para GPU

Nesta seção é apresentado o gerador de código para o algoritmo *K-means* para GPU Nvidia em CUDA. O gerador é parametrizável em função do número de dimensões n e do número de centroide k. Cada *thread* é responsável pela classificação de um ponto e executa os seguintes passos: o cálculo da distância em relação a todos os centroides, a redução de soma e a redução para a determinação do centroide mais próximo. Para evitar divergência com condicionais em GPU, foram usadas condicionais simples. A etapa de reposicionamento também é realizada na GPU.

Ao iniciar o cálculo concorrente de milhares de *threads*, a GPU mascara a latência da leitura na memória e das dependências de dados no nível de instruções de todos os passos da etapa de classificação, incluindo o mapeamento e as duas reduções. O gargalo é a etapa final de reposicionamento que envolve a redução com um acumulador para cada dimensão de cada centroide. Duas versões foram propostas para a fase de reposicionamento dos centroides. Uma baseada nas operações atômicas que possuem suporte em hardware nas novas GPUs (a partir da geração Kepler) e outra baseada em memória compartilhada, com segmentação a partir de blocos e bancos distintos de memória compartilhada em função dos valores de n e k. A Figura 4 ilustra o código gerado para o exemplo com duas dimensões e dois centroides na versão mais simples baseada em operações atômicas.

```
__global__ void kmeans(int *in, int *c, int *nC,
                       int *total, const int n) {
 int i;
 i = (blockIdx.x * blockDim.x + threadIdx.x) * DIM
     ;

 // leitura do Ponto
 int pd0 = in[i + 0];
 int pd1 = in[i + 1];

 // Map para distancia
 int k0d0 = pd0 - c[0]; // C0_0
 int k0d1 = pd1 - c[1]; // C0_1
 int k1d0 = pd0 - c[2]; // C1_0
 int k1d1 = pd1 - c[3]; // C1_1

 // Quadrado Distancia
 k0d0 *= k0d0; k0d1 *= k0d1;
 k1d0 *= k1d0; k1d1 *= k1d1;

 // Reducao de Soma para Distancia
 k0d0 = k0d0+k0d1;// Distancia de C0
 k1d0 = k1d0+k1d1;// Distancia de C1

 // Reducao de Minimo
 int minId;
 minId = (k1d0 < k0d0) ? 1 : 0;

 // Inclusao do Ponto para Reposicionamento
 atomicAdd(&(nC[DIM*minId+0]),pd0);
 atomicAdd(&(nC[DIM*minId+1]),pd1);
 atomicAdd(&(total[minId]), 1);
}
```

Figure 4. Código simplificado para *K-means* gerado para GPU com $k = 2$ e $n = 2$ realizando a redução de reposicionamento com operações atômicas.

VI. Experimentos e Resultados

Para avaliar os geradores propostos foi usada a base de dados *US Census 1990* [13]. Esta base de dados também foi usada em outros trabalhos com FPGA e GPU [14], [15], [6]. O tamanho da palavra utilizado para cada dimensão foi de 16 bits e a base tem 2 milhões de pontos. Para avaliar a variação do desempenho em função do número de centroides e do número de dimensões, utilizamos combinações valores de n e k variando na faixa 2, 4, 8, 16 e 32.

Os resultados foram comparados com a execução em uma CPU Intel Xeon E5-2630 v3 @ 2.40GHz com uma implementação do *K-means* em C, semelhante ao código gerado para GPU, compilado com GCC 4.8.4, com a opção -O3, executando como uma *thread*. O tempo de execução foi medido com a utilização de *chrono::high_resolution_clock*. Além disso, três outras implementações foram utilizadas como referência. A primeira é a implementação popular para execução do *K-means* encontrada no pacote *Scikit-learn* [16]. A segunda é a implementação em OpenMP com 16 *threads* disponibilizada pelo pacote Rodinia, versão 3.1 [17], [18]. A terceira é a implementação em GPU/CUDA também do pacote Rodinia, versão 3.1, executada na GPU K40. O código CPU, usado como referência, foi em média 6,5 vezes mais rápido que a implementação *Scikit-learn*. A versão CPU foi em média 3,42 e 6,83 vezes mais lenta que as implementações Rodinia em OpenMP e GPU/CUDA, respectivamente. A plataforma Intel HARP v.2 possui um FPGA Arria 10 modelo 10AX115U3F45E2SGE3. Duas GPUs foram utilizadas, uma Nvidia Tesla K40c com 12GB de memória RAM e uma frequência de relógio de 745 MHz e uma Nvidia Pascal 1080ti com 11 GB e uma frequência de relógio de 1.584 GHz. Todos os experimentos foram conduzidos com 10 repetições e são apresentados os resultados médios.

A Figura 5 mostra os resultados do ganho de aceleração agrupados pelo número de dimensões. Pode-se perceber que o ganho de aceleração das GPUs e do FPGA em relação a CPU para conjunto de pontos com a mesma quantidade de dimensões aumenta em função de k, o motivo é o reúso dos dados de $3k - \frac{1}{n}$ operações por byte. Para cada grupo avaliado, as três primeiras barras são os resultados do *scikit-learn*, Rodinia OpenMP e GPU. As três últimas barras são os resultados dos geradores de código propostos para o *K-means*: FPGA, K40 e 1080ti. Os tempos de execução foram normalizados em relação ao tempo de execução da CPU, usado como referência. Quanto maior o valor, melhor é o resultado. A linha em negrito com o valor 1 simboliza o tempo de execução na CPU. As GPUs K40 e 1080ti chegam a ser até 54 e 67 vezes mais rápida que a CPU. O FPGA foi até 13,85 mais rápido. Pode-se observar que a implementação padrão *scikit-learn* é, em média, 63 vezes mais lenta que o FPGA, 245,5 e 250 mais lenta que as GPUs K40 e 1080ti, respectivamente. Em relação a implementação Rodinia para GPU, os geradores para a K40 e a 1080ti foram, em média, de 3 e 4 vezes mais rápidos, respectivamente. Em relação a versão Rodinia OpenMP com 16 *threads*, os geradores com a K40 e a 1080ti foram, em média, de 6 e 8 vezes mais rápidos.

O maior limitador da plataforma Intel HARP v.2 é a leitura de dados. Como o FPGA não tem memória local, todas as leituras são feitas da memória da CPU a cada iteração. Além disso, para $k = 2$ com 32 dimensões, o conjunto de dados é 16 vezes maior que para 2 dimensões. Na GPU, os dados são transferidos uma vez pelo barramento PCI e permanecem na memória global da GPU até o fim de todas as iterações. A memória global das GPUs tem uma vazão de 200-400 GB/s, o que é significativamente maior que transferência CPU-FPGA, que fica em torno de 16 GB/s. O tempo de execução inclui todas as transferências de dados. É importante ressaltar que o HARP v.2 é um protótipo que pode não refletir o desempenho de sistemas futuros da Intel. Recém lançado em maio de 2018, o processador Intel Xeon 6138P traz integrado um FPGA Arria 10 [19] e pode atingir taxas de transferência de 160 GB/s, 10 vezes mais que o HARP v.2.

O potencial de eficiência energética do FPGA é outro ponto importante. A plataforma HARP v.2 tem um consumo médio de 22,351 Watts aferidos durante as execuções com a

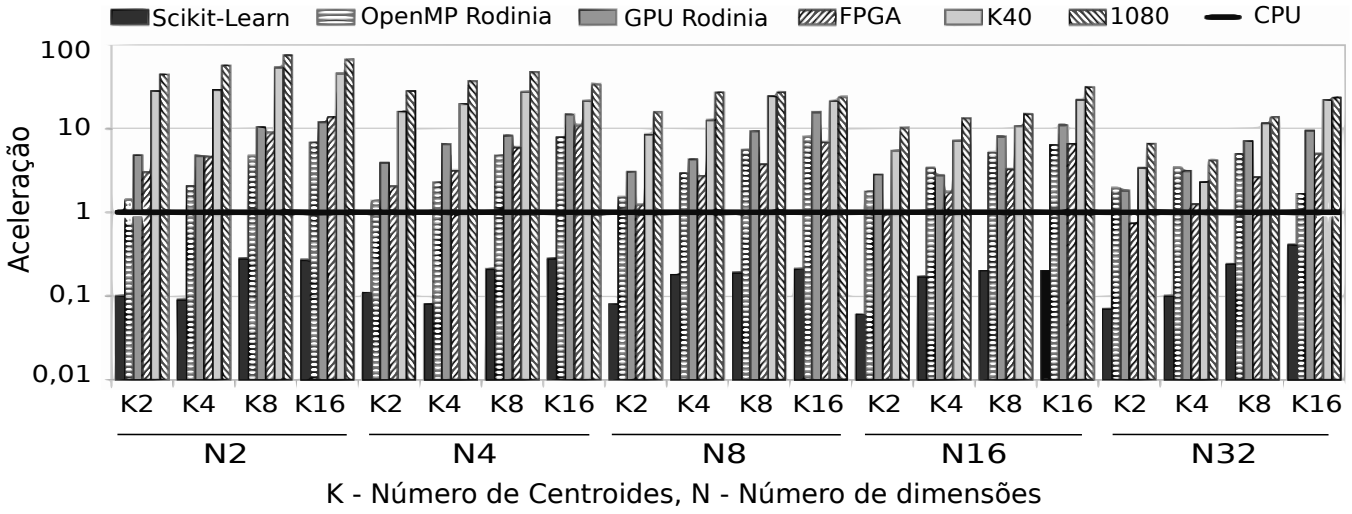

Figure 5. Acelerações das GPUs K40 e 1080 e do FPGA normalizadas em relação ao tempo de execução com um *thread* em CPU. Escala logarítmica.

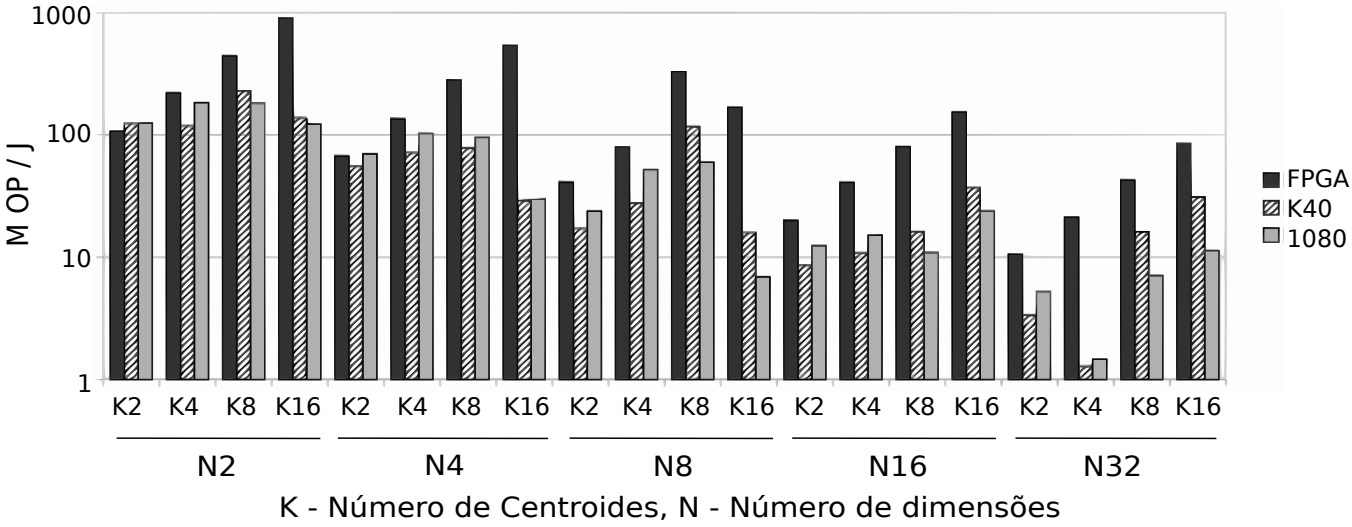

Figure 6. Eficiência energética para a execução dos *kernels* em GPU e em FPGA.

ferramenta disponibilizada pela API OPAE da Intel. Como referência para comparação foi utilizada uma estimativa de potência aferida para GPU K40 disponibilizada em [20] de 134 Watts. A GPU 1080ti tem o consumo de 300 Watts. Com base nestes dados, a Figura 6 apresenta a eficiência energética como a quantidade de milhões de operações (MOPs) por Joule obtidos em cada experimento para uma iteração nas GPUs e no FPGA. Para um mesmo valor de n, quanto maior o reúso, mais eficiente é a solução. Pode-se observar que a solução com FPGA é promissora, mesmo sendo um protótipo da Intel com limitações de leitura de memória. Este fato também corrobora o interesse das grandes empresas nas soluções com FPGA em nuvem, como por exemplo a Microsoft [10] e a Amazon [9].

Como já mencionado, o HARP v.2 é ainda um protótipo. Ao realizar a chamada do acelerador, a sobrecarga de tempo da API Intel domina o tempo de execução. A Figura 7 exibe o tempo de execução da chamada do *kernel* do acelerador para cálculo de uma iteração no FPGA incluindo a transferência dos dados, mas não considera o tempo gasto pela API de software da Intel para a realização da inicialização entre as chamadas sucessivas do acelerador (entre as iterações). Em média, o tempo da execução, incluindo a transferência de dados, é de apenas 30% do tempo da iteração. A sobrecarga das APIs é aproximadamente de 70% do tempo da iteração. Para GPU, a sobrecarga das chamadas é pequena, onde o tempo da execução e da transferência é da ordem de 95% ou mais. Ao desconsiderar a sobrecarga da API, o FPGA foi mais rápido que a GPU K40 em três casos, apesar de ter uma taxa de leitura de memória 13,75

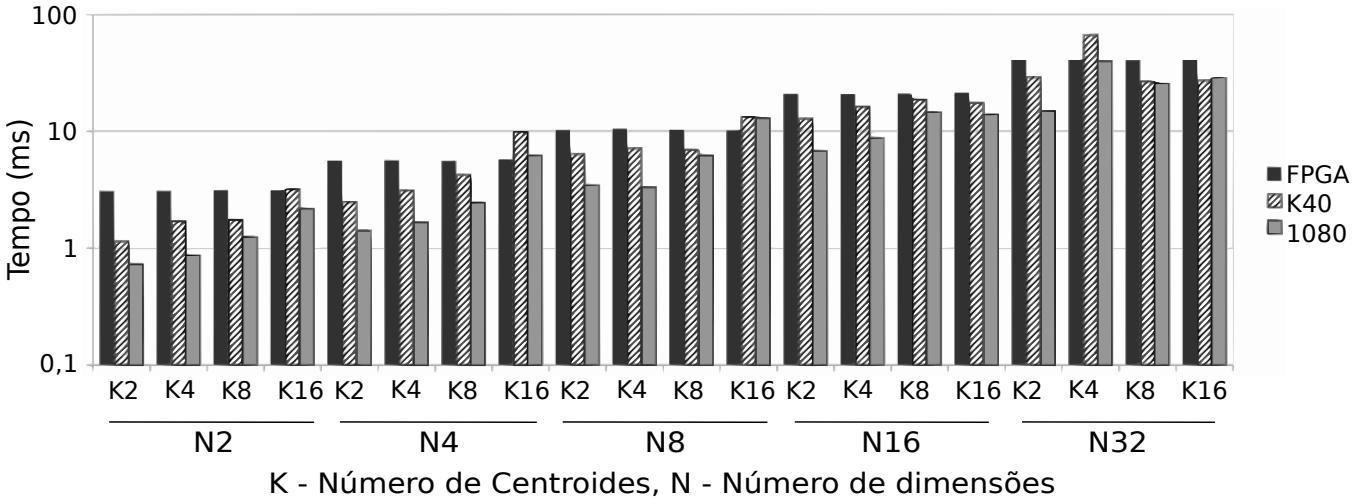

Figure 7. Tempos de execução da chamada dos aceleradores.

vezes menor e uma frequência de relógio de 200 MHz em comparação com os 745 MHz da K40. Já a GPU 1080ti é mais rápida pois possui uma frequência de relógio de 1,58 GHz e uma taxa de leitura de memória de 485 GB/s.

Vale ressaltar uma diferença da Figura 7, na qual o tempo não varia em função de k para um dado n, do comportamento dos resultados anteriores. Nas Figuras 5 e 6, a eficiência energética e a aceleração aumentam em função de k. Na Figura 5, a medida é relativa ao tempo da CPU, portanto apesar do tempo de execução do FPGA não variar para mesmo n, o tempo da CPU varia, aumentando em função de k pois não tira proveito do paralelismo de calcular todas as distâncias ao mesmo tempo. No caso da Figura 6, como número de operações aumenta em função de k, a eficiência aumenta, uma vez que o tempo de execução não se altera para um dado n. Ao dobrar o valor de n dobra-se a quantidade de dados. Por exemplo, para 2 milhões de pontos e $n = 2$, tem-se 8 MB de dados e para $n = 4$ teremos 16 MB. Outra observação é a redução da quantidade de cópias do algoritmo por linha de cache com o aumento da quantidade de dimensões. A linha de *cache* do HARP v.2 é de 64 Bytes. Com $n = 2$, cada ponto consome 4 bytes e pode-se instanciar 16 cópias que processam 16 pontos em paralelo. Para $n = 4$, cada ponto terá 8 bytes e tem-se 8 cópias do processo por linha de *cache*.

Ao analisar o aumento de k para um n fixo, pode-se observar que com mais centroides, há um maior reúso dos dados. Para $k = 2$ e $n = 2$, a Figura 8(a) mostra que existem 11 operações por ponto e com $k = 4$, na Figura 8(b), são 23 operações por ponto. Desse modo, o mesmo ponto será comparado com 4 centroides simultaneamente, o que explora o paralelismo e aumenta o reúso.

A Figura 9 apresenta a quantidade de recursos do FPGA gastos em função dos valores de k e n, que é um aspecto

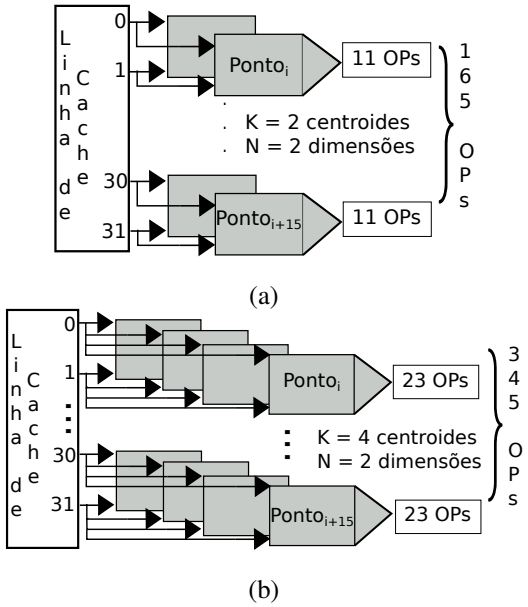

Figure 8. Impacto do valor de k: (a) 165 Operações por 64 bytes; (b) 345 ops.

favorável aos FPGAs. São apresentadas apenas a quantidade de ALMs (blocos lógicos) e de DSPs (unidades lógica-aritméticas), pois o número de módulos de memória não varia, uma vez que são utilizados apenas pela interface com a CPU. As ALMs são usadas para construir a interface, controle, operações de soma e redução de mínimo. Em média, 18-20% das ALMs são usadas pela interface da Intel e apenas 2-6% das ALMs são gastas pelo gerador para implementar o controle e as operações de soma e redução. O maior valor de k é apenas limitado pela quantidade de DSPs disponíveis no FPGA utilizado. Entretanto, existe uma perspectiva favorável aos FPGAs, pois o novo FPGA Stratix

X da Intel possui 6760 DSPs [21] em comparação com os 1518 do Arria 10 do HARP v.2. Além do aumento da quantidade de DSP, a flexibilidade do FPGA permite que futuras extensões do gerador façam uso de aritmética dedicada com ALMs (que são abundantes) para o cálculo do quadrado das distâncias ao invés dos DSPs como multiplicadores. Portanto, com vazão de dados, mais DSPs, o desempenho do FPGA para o *K-means* pode aumentar significativamente.

VII. Trabalhos Relacionados

Esta seção apresenta um estudo das implementações do *K-means* em FPGAs e GPUs. Um gerador parametrizável em função do número de dimensões e centroides para FPGA foi proposto em [22]. Entretanto, o artigo não apresenta resultados de execução, apenas estimativas e simulação. É importante ressaltar que uma descrição que pode ser simulada em FPGA, não necessariamente irá executar corretamente. Além disso, o tempo de sincronização, chamada, leitura e escrita de dados pode não estar incluso. Uma estimativa para execução do *K-means* em FPGA é apresentada em [23], também sem resultados concretos com execução. Outra implementação em uma plataforma CPU/FPGA Xilinx é apresentada em [1]. A aceleração é estimada em função do potencial de aceleração do FPGA e da CPU, inclusive a comunicação, o que mostra um potencial de aceleração de até 150x para o *K-means*, porém o sistema não foi avaliado em tempo de execução. Uma abordagem em alto nível para FPGA com a utilização da linguagem OpenCL é apresentada em [3], na qual a implementação utiliza uma função para a classificação e outra para o reposicionamento dos centroides. A classificação utiliza a distância de Manhattan no lugar da distância Euclidiana. Esta proposta apresenta melhora de desempenho apenas para valores grandes de k, maiores que 128. Em [24], um acelerador para FPGA foi implementado, onde os parâmetros k, n e número de iterações podem ser alterados em tempo de execução. Porém o formato dos dados é fixo (64 bits), o valor máximo está limitado a apenas 16K pontos e foi apenas prototipado em uma placa de baixa custo com cálculo paralelo de apenas 2 pontos por vez, o que restringe seu uso em problemas reais. Para $k = 8$ e $n = 4$, o desempenho é de apenas 6M operações/s a 50 MHz, o gerador proposto neste trabalho executa 9,7G operações/s à 200 MHz, ou seja, 250 vezes mais rápido que o acelerador proposto por [24] e não está limitado a 16K entradas.

Como não existe uma padronização, é difícil realizar uma comparação entre as implementações. A comparação será feita através de bases de dados de mesmo tamanho e considerando o tempo de uma iteração no qual o número de operações realizadas é equivalente. A arquitetura FPGA proposta em [14] é baseada no paradigma *Map-Reduce* onde a avaliação foi feita com *benchmarks* de apenas 2 e 4 dimensões para $k = 4$. Ao comparar com os resultados apresentados na Seção VI, o gerador proposto neste trabalho apresenta tempos de execução duas vezes mais rápidos que resultados de [14] para $n = 2$ e 4. Ademais, o gerador é genérico e não foi dedicado a dois valores de n. Uma arquitetura com elementos de processamento para o *K-means* em FPGA foi proposta em [25], que é pelo menos duas vezes mais lenta que o gerador proposto neste trabalho para $k = 3$ e $n = 10$. O trabalho apresentado em [6] é o mais próximo da abordagem proposta em termos da plataforma de FPGA. Os resultados também são avaliados em tempo de execução. O FPGA é o protótipo da HARP v.1, já descontinuado. O FPGA é usado para calcular a etapa de classificação e a CPU calcula o reposicionamento. Apenas $k = 8, n = 2$ e $k = 4, n = 4$ são avaliados para os quais a abordagem proposta neste trabalho em FPGA é 20 vezes ou mais rápida que o resultado apresentado em [6]. Uma biblioteca para GPU é apresentada em [26], porém os resultados são pelo menos 10 vezes mais lentos da solução em FPGA e na GPU K40 apresentada nesse trabalho. Em relação a implementação proposta para GPU 1080ti, os resultados do trabalho proposto em [26] foram 50 vezes mais lentos. O maior problema é que parte da classificação e reposicionamento são implementados na CPU. Uma comparação entre as soluções com CPU, GPU e FPGA foi apresentada em [27], onde a solução com FPGA foi a mais rápida executando na Nuvem da Amazon [9]. A solução apresentada por este trabalho em FPGA é genérica com resultados equivalentes em desempenho considerando $k = 8$ e $n = 5$ aos apresentados para uma abordagem específica em [27]. Como já apresentado na Figura 5, a implementação Rodinia para GPU [18] foi de 3 à 4 vezes mais lenta que o gerador proposto neste trabalho.

VIII. Conclusão

Este trabalho apresenta um gerador parametrizável do algoritmo *K-means* para FPGAs e GPUs. O desempenho é maximizado ao se especializar em um domínio específico e explorar o paralelismo espacial e temporal. Apesar de ser específico, existe uma demanda por agrupamento de dados, onde o *K-means* e suas variações têm muitas aplicações. Os resultados experimentais mostraram ganho em tempo de execução em comparação a CPU de até 54,55 e 13,85 vezes para as GPUs k40 e 1080ti e a plataforma HARP v.2, respectivamente. O gerador executou com uma frequência de 200 MHz no FPGA, mas pode ser ajustado para executar a 400 MHz, que dobrará seu desempenho. Apesar do desempenho do FPGA ter sido limitado pelas baixas taxas de transmissão de 16 GB/s na leitura de memória do protótipo HARP v.2 da Intel, existe um potencial para os novas plataformas FPGAs da Intel e da Xilinx onde o desempenho da transferência de dados deve melhorar significativamente.

Como trabalhos futuros, há dois aspectos importantes do algoritmo *K-means* que podem ser explorados: a determinação da quantidade de centroides e a seleção de um subconjunto relevante de atributos. Essas duas funcionalidades podem ser adicionadas de forma eficiente

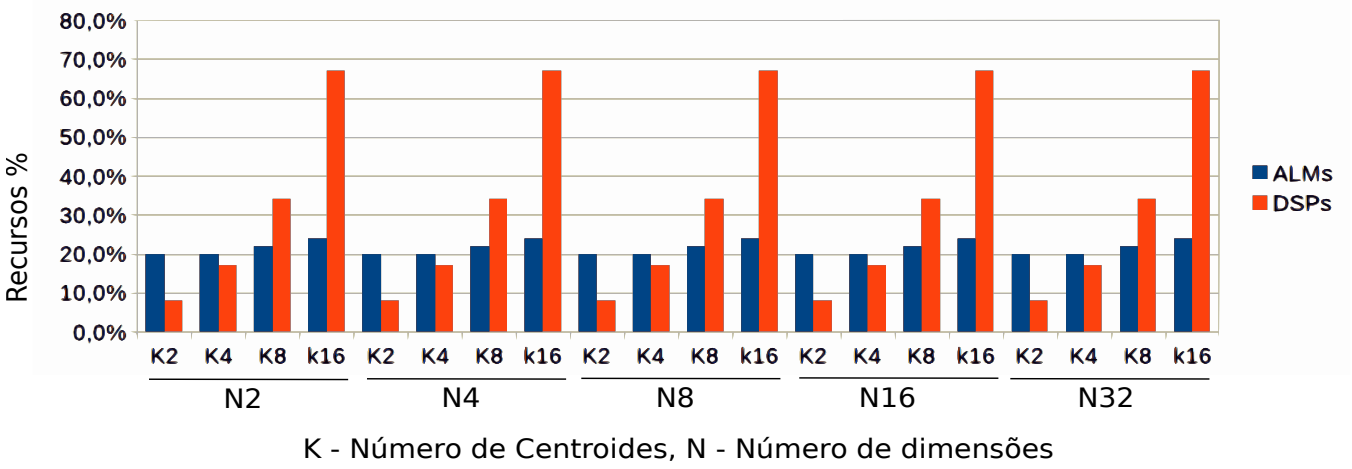

Figure 9. Recursos do FPGA utilizados pelos aceleradores.

às implementações aqui propostas, tanto em GPU quanto FPGA, uma vez aumentarão ainda mais o reúso de dados. Outra possibilidade é a computação aproximada que vem sendo explorada em vários algoritmos de aprendizado de máquina. Pode-se esperar que a flexibilidade do FPGAs tenha um nicho melhor neste domínio que as GPUs e as CPUs. Para a nova geração *Volta* de GPUs da Nvidia [28], existe ainda um potencial a ser explorado pelo *K-means* com os operadores TPU que fazem multiplicação e acúmulo de matrizes com estruturas sistólicas. Com relação às CPUs, o *K-means* pode explorar paralelismo a nível dos múltiplos núcleos e também o paralelismo no nível de instrução com as extensões vetoriais AVX.

Agradecimentos

Este estudo foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Código Financeiro 001. Esta pesquisa também foi apoiada pela FAPEMIG, CNPq, FUNARBE, Intel, Nvidia, Universität Paderborn pela disponibilização dos equipamentos e, finalmente, pela Synopsys por seu suporte e licenças do software acadêmico.

References

[1] K. Neshatpour *et al.*, "Energy-Efficient Acceleration of Big Data Analytics Applications Using FPGA," in *IEEE International Conference on Big Data*, 2015.

[2] ACM, "A.M. Turing Award," in *ACM Award 2017 https://amturing.acm.org/*, 2018.

[3] Q. Y. Tang and M. A. Khalid, "Acceleration of K-Means Algorithm Using Altera SDK for OpenCL," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 10, no. 1, p. 6, 2016.

[4] E. W. Forgy, "Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications," *Biometrics*, vol. 21, pp. 768–769, 1965.

[5] Y.-M. Choi and H. K.-H. So, "Map-Reduce Processing of K-Means Algorithm with FPGA-Accelerated Computer Cluster," in *IEEE International Conference ASAP*, 2014.

[6] T. S. Abdelrahman, "Accelerating K-Means Clustering on a Tightly-Coupled Processor-FPGA Heterogeneous System," in *IEEE International Conference ASAP*, 2016.

[7] M. J. Zaki and W. Meira Jr, *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, 2014.

[8] P. Gupta, "Accelerating Datacenter Workloads," in *Field Programmable Logic and Applications, Keynote - Slides available at www.fpl2016.org*, 2016.

[9] Amazon, "Amazon AWS," in *Elastic Compute Cloud - Amazon EC2 - AWS http://aws.amazon.com/ec2/*, 2018.

[10] A. M. e. a. Caulfield, "A Cloud-Scale Scceleration Architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[11] L. Bragança, F. Alves, J. C. Penha, G. Coimbra, R. Ferreira, and J. A. M. Nacif, "Simplifying HW/SW Integration to Deploy Multiple Accelerators for CPU-FPGA Heterogeneous Platforms," in *IEEE SAMOS*, 2018.

[12] S. Takamaeda-Yamazaki, "Pyverilog: A Python-Based Hardware Design Processing Toolkit for Verilog HDL," in *Applied Reconfigurable Computing*, Apr 2015.

[13] D. Dheeru and E. Karra Taniskidou, "UC Irvine Machine Learning Repository," in *UCI Machine Learning Repository http://archive.ics.uci.edu/ml*, 2017.

[14] Z. Li, J. Jin, and L. Wang, "High-Performance K-Means Implementation Based on a Simplified Map-Reduce Architecture," *arXiv preprint arXiv:1610.05601*, 2016.

[15] R. Kaplan, L. Yavits, and R. Ginosar, "Prins: Processing-in-Storage Acceleration of Machine Learning," *IEEE Transactions on Nanotechnology*, 2018.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, Michel *et al.*, "Scikit-Learn: Machine Learning in Python," *Journal of machine learning research*, vol. 12, 2011.

[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IEEE Int. Symposium on Workload Characterization*, 2009.

[18] Rodinia, "Rodinia: Accelerating Compute-Intensive Applications with Accelerators," in *Version 3.1 http://lava.cs.virginia.edu/Rodinia/download_links.htm*, 2016.

[19] Intel, in *Introducing the Intel Xeon Scalable processor with integrated Arria 10 FPGA https://itpeernetwork.intel.com/intel-processors-fpga-better-together/*, 2018.

[20] M. Ferro *et al.*, "Analysis of GPU Power Consumption Using Internal Sensors," in *Workshop em Desempenho de Sistemas Computacionais e de Comunicacao*, 2017.

[21] Stratix, in *Device Overview https://www.altera.com/products/fpga/stratix-series/*, 2018.

[22] A. Amaricai, "Design Trade-offs in Configurable FPGA Architectures for K-Means Clustering," *Studies in Informatics and Control*, vol. 26, no. 1, pp. 43–48, 2017.

[23] K. Neshatpour *et al.*, "Big Biomedical Image Processing Hardware Acceleration: A Case Study for K-Means and Image Filtering," in *Int. Symposium on Circuits and Systems*, 2016.

[24] L. A. Maciel, M. A. Souza, and H. C. de Freitas, "Projeto e Avaliação de uma Arquitetura do Algoritmo de Clusterização k-means em vhdl e fpga," *WSCAD*, 2017.

[25] D. Lee, A. Althoff, D. Richmond, and R. Kastner, "A Streaming Clustering Approach Using a Heterogeneous System for Big Data Analysis," in *IEEE/ACM ICCAD*, 2017.

[26] J. Bhimani, M. Leeser, and N. Mi, "Accelerating K-Means Clustering with Parallel Implementations and GPU Computing," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 2015.

[27] K. Huang, "K-Means Parallelism on FPGA," Master's thesis, Northeastern University, Boston, USA, 2017.

[28] N. P. Jouppi, *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *International Symposium on Computer Architecture (ISCA)*, 2017.