P2.K-means on FPGA via OpenCL

Nuno Paulino
CTM
INESC TEC

Overview

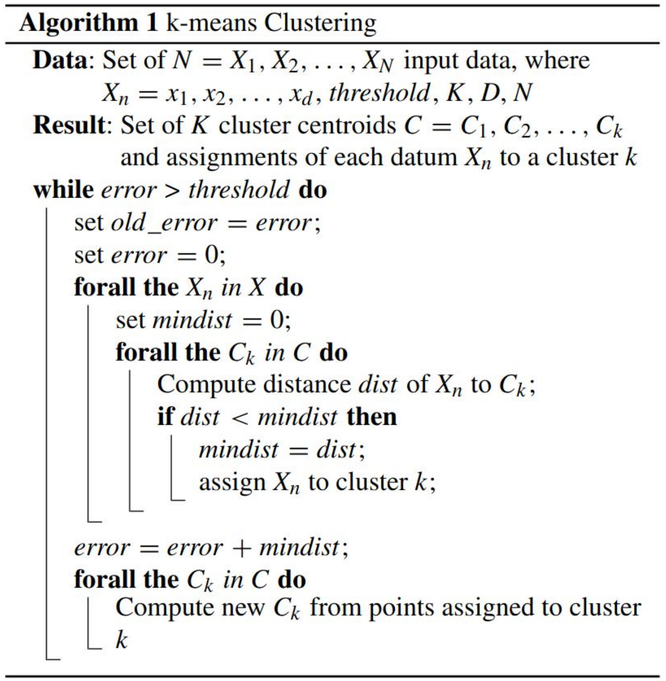
- Title: *Optimizing OpenCL Code for Performance on FPGA: k-Means Case Study With Integer Data Sets (https://ieeexplore.ieee.org/document/9170625)*
- Journal: *IEEE Access*
    - Q1 in Computer Science
    - Impact Factor: 3.37 (@2020)
    - Approx. 6 citations per paper
- 21 Page paper
    - Submitted: 28 Jul. 2020 → Accepted: 2 Aug. 2020!
- Project: *PEPCC (Power Efficiency and Performance for Embedded and HPC Systems with Custom CGRAs)*
- Keywords: ***OpenCL, k-means, clustering, FPGA, hardware accelerator, HLS***

Objectives

- Study a use case of HLS for FPGAs using OpenCL
- Outperform a sequential CPU execution of k-means
  - When executing k-means as C on CPU
  - When executing k-means as OpenCL kernel on CPU
- Compare runtime, power consumption, and power/performance tradeoff

k-means

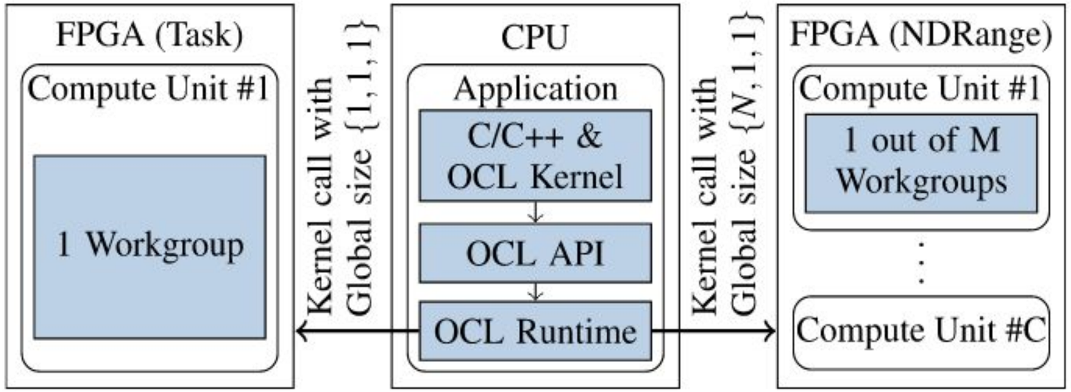
- From a given set of initial cluster centroids:
    a. for each point, compute distance to all centroids
    b. assign each point to its closest centroid
    c. compute new centroids based on point assignments
    d. repeat from “a” until centroids converge (to a given tolerance)
- What is the best way to parallelize?



Algorithm 1 k-means Clustering

```
Data: Set of $N = X_1, X_2, \ldots, X_N$ input data, where
      $X_n = x_1, x_2, \ldots, x_d$, $threshold$, $K$, $D$, $N$
Result: Set of $K$ cluster centroids $C = C_1, C_2, \ldots, C_k$
        and assignments of each datum $X_n$ to a cluster $k$
while $error > threshold$ do
    set $old\_error = error$;
    set $error = 0$;
    forall the $X_n$ in $X$ do
        set $mindist = 0$;
        forall the $C_k$ in $C$ do
            Compute distance $dist$ of $X_n$ to $C_k$;
            if $dist < mindist$ then
                $mindist = dist$;
                assign $X_n$ to cluster $k$;
    $error = error + mindist$;
    forall the $C_k$ in $C$ do
        Compute new $C_k$ from points assigned to cluster
        $k$
```

OpenCL Workgroup Computing Model

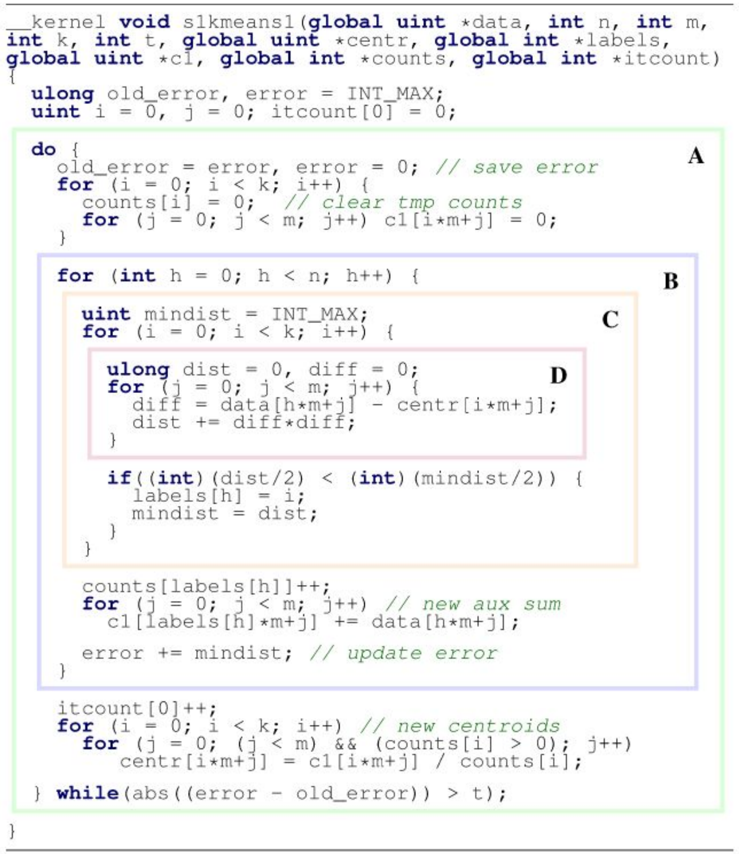


OpenCL Task-Kernel vs NDRange Kernel execution; for NDRange, workgroups have local size {1 < n < N, 1, 1}, where N = total # workitems

Baseline OpenCL

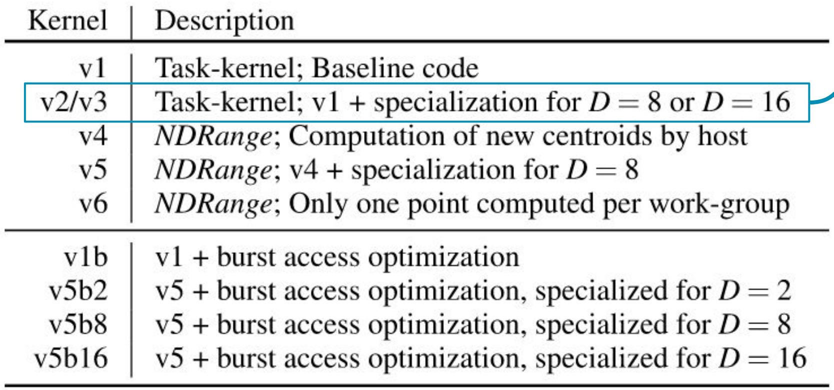
- Straight C → OpenCL conversion
- Purely sequential
    - In OpenCL, its classified as a **“task-kernel”**
    - Does not exploit workgroup model

- In this case
    - FPGA can explore deep hardware pipelining, where CPU cannot
    - **One compute unit is instantiated on the FPGA**



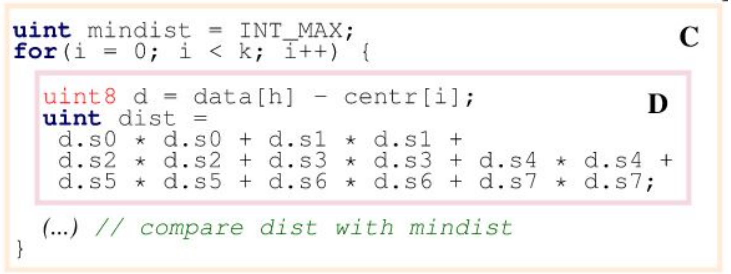
```
__kernel void s1kmeans1(global uint *data, int n, int m,
int k, int t, global uint *centr, global int *labels,
global uint *c1, global int *counts, global int *itcount)
{
  ulong old_error, error = INT_MAX;
  uint i = 0, j = 0; itcount[0] = 0;

  do {
    old_error = error, error = 0; // save error
    for (i = 0; i < k; i++) {
      counts[i] = 0;  // clear tmp counts
      for (j = 0; j < m; j++) c1[i*m+j] = 0;
    }

    for (int h = 0; h < n; h++) {

      uint mindist = INT_MAX;
      for (i = 0; i < k; i++) {

        ulong dist = 0, diff = 0;
        for (j = 0; j < m; j++) {
          diff = data[h*m+j] - centr[i*m+j];
          dist += diff*diff;
        }

        if((int)(dist/2) < (int)(mindist/2)) {
          labels[h] = i;
          mindist = dist;
        }
      }

      counts[labels[h]]++;
      for (j = 0; j < m; j++) // new aux sum
        c1[labels[h]*m+j] += data[h*m+j];

      error += mindist; // update error
    }

    itcount[0]++;
    for (i = 0; i < k; i++) // new centroids
      for (j = 0; (j < m) && (counts[i] > 0); j++)
         centr[i*m+j] = c1[i*m+j] / counts[i];

  } while(abs((error - old_error)) > t);
}
```

Optimizations



Kernel	Description
v1	Task-kernel; Baseline code
v2/v3	Task-kernel; v1 + specialization for $D = 8$ or $D = 16$
v4	*NDRange*; Computation of new centroids by host
v5	*NDRange*; v4 + specialization for $D = 8$
v6	*NDRange*; Only one point computed per work-group
v1b	v1 + burst access optimization
v5b2	v5 + burst access optimization, specialized for $D = 2$
v5b8	v5 + burst access optimization, specialized for $D = 8$
v5b16	v5 + burst access optimization, specialized for $D = 16$

Different tested k-means kernel versions



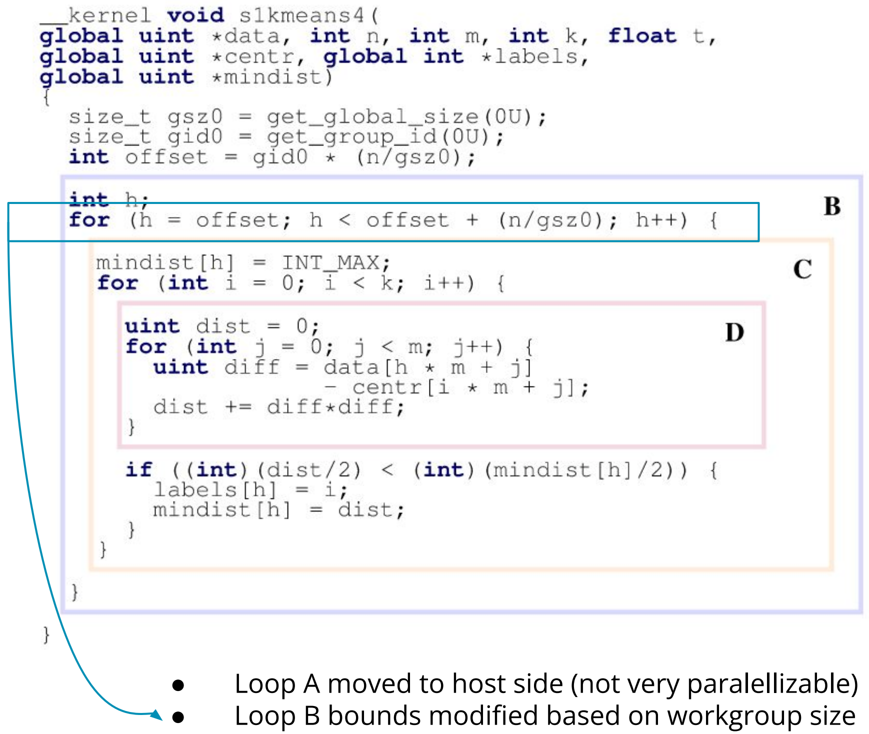
```
uint mindist = INT_MAX;                                          C
for(i = 0; i < k; i++) {

  uint8 d = data[h] - centr[i];                                 D
  uint dist =
   d.s0 * d.s0 + d.s1 * d.s1 +
   d.s2 * d.s2 + d.s3 * d.s3 + d.s4 * d.s4 +
   d.s5 * d.s5 + d.s6 * d.s6 + d.s7 * d.s7;

  (...) // compare dist with mindist
}
```

Excerpt from v2
Removal of one inner w/ 8 iterations loop using a vector datatype of 8 elements

- In this case
  - Vectorization removes on inner loop
  - We confirmed that Intel's OpenCL runtime performs auto-vectorization

Optimizations - v4/v5

- Workgroup model
  - “Normal” for OpenCL workloads
  - Nr workgroups determined by max. workgroup size and total nr. of workgroups
  - Workgroups → parallel
- In this case
  - CPU explores parallel work groups due to independent data
  - But FPGA can **in addition** explore pipelining of inner loops
  - **Multiple compute units are instantiated on the FPGA**

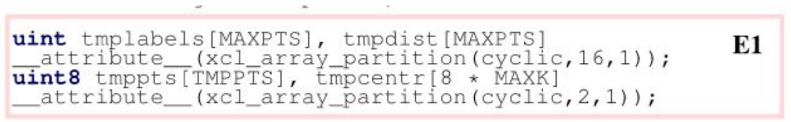


```
__kernel void s1kmeans4(
global uint *data, int n, int m, int k, float t,
global uint *centr, global int *labels,
global uint *mindist)
{
  size_t gsz0 = get_global_size(0U);
  size_t gid0 = get_group_id(0U);
  int offset = gid0 * (n/gsz0);

  int h;
  for (h = offset; h < offset + (n/gsz0); h++) {

    mindist[h] = INT_MAX;
    for (int i = 0; i < k; i++) {

      uint dist = 0;
      for (int j = 0; j < m; j++) {
        uint diff = data[h * m + j]
                  - centr[i * m + j];
        dist += diff*diff;
      }

      if ((int)(dist/2) < (int)(mindist[h]/2)) {
        labels[h] = i;
        mindist[h] = dist;
      }
    }

  }

}
```

- Loop A moved to host side (not very paralellizable)
- Loop B bounds modified based on workgroup size

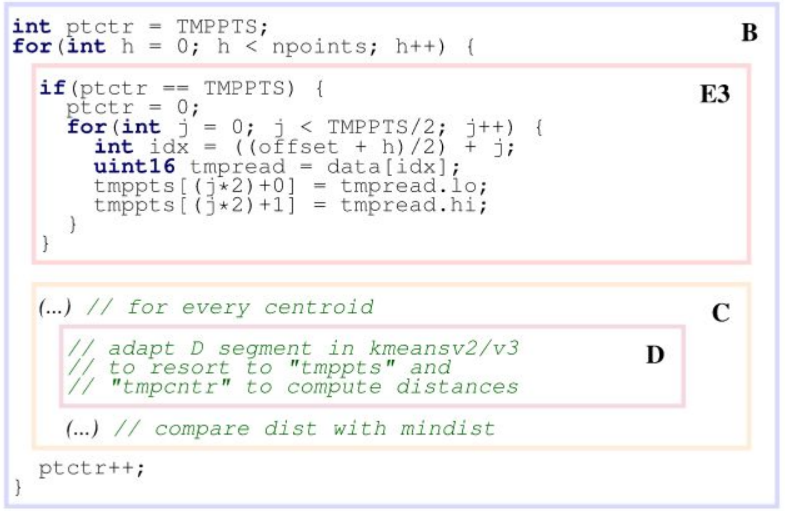
Optimizations - v5b

- Workgroup model with burst memory access inference
    - **Loop E3 - Burst read points**
    - Uses more device BRAM
    - Explicit local multi-port memories load up to TMPPTS points
        - *TMPPTS* could have been larger, up to device limits



```
uint tmplabels[MAXPTS], tmpdist[MAXPTS]
__attribute__(xcl_array_partition(cyclic,16,1));
uint8 tmppts[TMPPTS], tmpcentr[8 * MAXK]
__attribute__(xcl_array_partition(cyclic,2,1));
```

(E2 - loop for burst reading into “tmpcentr” omitted)



```
int ptctr = TMPPTS;
for(int h = 0; h < npoints; h++) {

  if(ptctr == TMPPTS) {
    ptctr = 0;
    for(int j = 0; j < TMPPTS/2; j++) {
      int idx = ((offset + h)/2) + j;
      uint16 tmpread = data[idx];
      tmppts[(j*2)+0] = tmpread.lo;
      tmppts[(j*2)+1] = tmpread.hi;
    }
  }

  (...) // for every centroid

    // adapt D segment in kmeansv2/v3
    // to resort to "tmppts" and
    // "tmpcntr" to compute distances

    (...) // compare dist with mindist

  ptctr++;
}
```

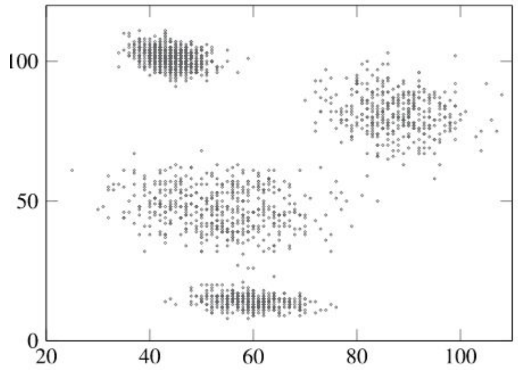
(E3 - loop for burst writing into outputs omitted)

Excerpt from v5b

Experimental Setup

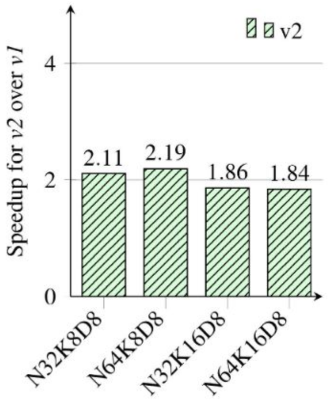
- Desktop CPU
  - Intel Core i7-6700K CPU (4 GHz)
  - Alpha Data ADM-PCIE-KU3
    - Kintex-6 XCKU060 FPGA
  - 32 GB RAM
- Execution
  - Host allocates input/output memory
  - Initial centroids computed using *kmeans++*
  - OpenCL API using Xilinx's runtime for FPGA target, or Intel's runtime for CPU
- Data
  - Generated synthetically by our own randomly correlated cluster generator

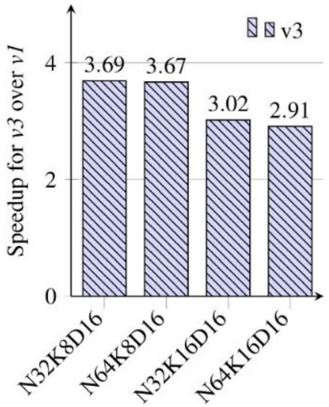
Alpha Data ADM-PCIE-KU3

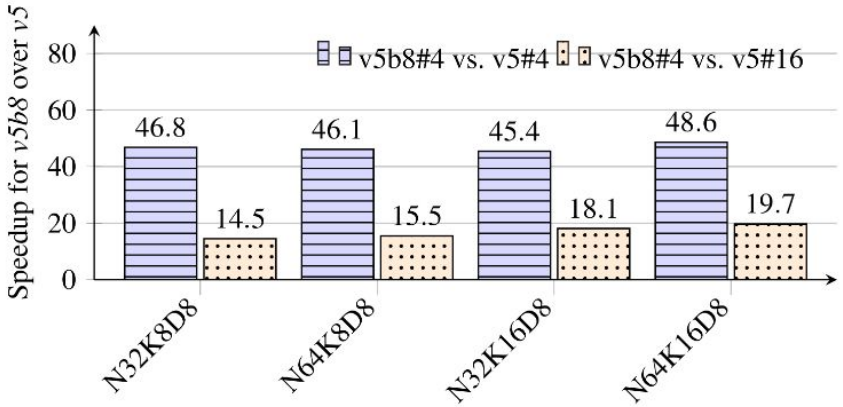


Example dataset generated for D = 2, K = 4, N = 4k

Experimental Results – Performance on FPGA







Speedup of vectorization alone vs OpenCL baseline (v1), on FPGA

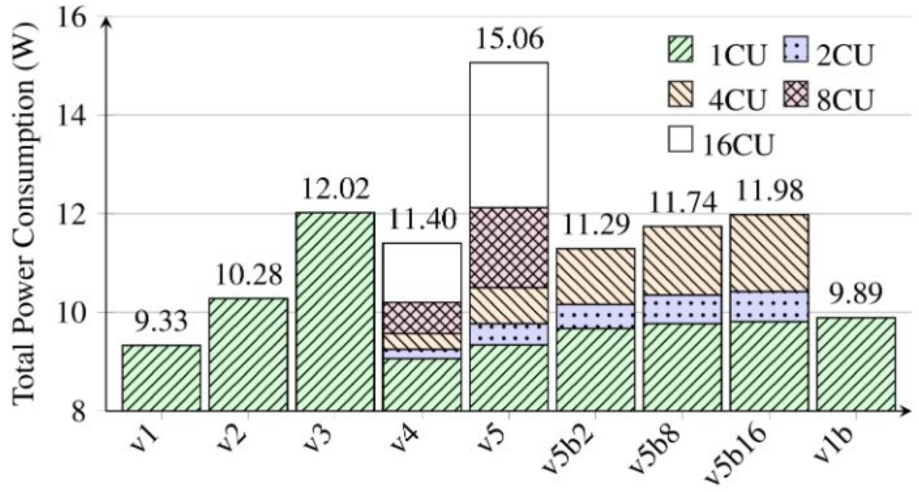
- i.e., task kernels w/ and w/o vectorization

Speedup of burst access over analogous versions (e.g., v5b over v5)

- Workgroup kernels w/ vectorization, w/ and w/o burst accesses

Experimental Results - Power on FPGA

- Power measured from post-route reports
    - For all code variants
    - For different numbers of compute units (where applicable)
- The best performing versions (v5b) only support up to 4 compute units
    - (Lack of FPGA resources)



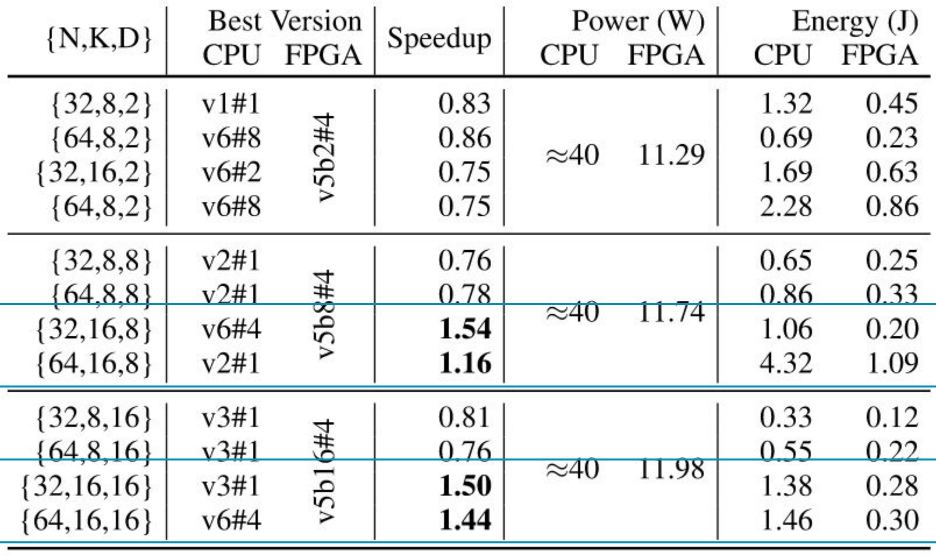
Power consumption on FPGA for all cases and different numbers of CUs

Experimental Results - Summary FPGA vs CPU

- Power measured on CPU using RAPL interface
- Compared best performant code version per device, per problem size

FPGA Wins!

FPGA Wins!



{N,K,D}	Best Version		Speedup	Power (W)		Energy (J)	
	CPU	FPGA		CPU	FPGA	CPU	FPGA
{32,8,2}	v1#1	v5b2#4	0.83	≈40	11.29	1.32	0.45
{64,8,2}	v6#8		0.86			0.69	0.23
{32,16,2}	v6#2		0.75			1.69	0.63
{64,8,2}	v6#8		0.75			2.28	0.86
{32,8,8}	v2#1	v5b8#4	0.76	≈40	11.74	0.65	0.25
{64,8,8}	v2#1		0.78			0.86	0.33
{32,16,8}	v6#4		**1.54**			1.06	0.20
{64,16,8}	v2#1		**1.16**			4.32	1.09
{32,8,16}	v3#1	v5b16#4	0.81	≈40	11.98	0.33	0.12
{64,8,16}	v3#1		0.76			0.55	0.22
{32,16,16}	v3#1		**1.50**			1.38	0.28
{64,16,16}	v6#4		**1.44**			1.46	0.30

Conclusions

- Mid-grade FPGA can outperform high-end CPU
    - Best version **725x faster** than OpenCL baseline on FPGA
    - But not without significant code re-factoring, producing non-portable OpenCL code
    - CPU still faster in most cases, but best FPGA case outperforms CPU **by 1.5x** with **4.8x** lesser power
- Four public artifacts
    - An Implementation of K-means written in C -
        - https://codeocean.com/capsule/3208075/tree/v1
    - A Test Harness for Multiple OpenCL Implementations of the k-means Algorithm
        - https://codeocean.com/capsule/2348736/tree/v1
    - A Generator of Randomly Correlated N-Dimentional Clusters
        - 10.13140/RG.2.2.34866.43200
    - A Batch of Integer Datasets for Clustering Algorithms
        - 10.21227/smta-vv06