Three is not a crowd

A CPU-GPU-FPGA K-means implementation

Borja Pérez Pavón	University of Cantabria
Carlos Escuín Blasco	Polytechnic University of Catalonia
Denisa-Andreea Constantinescu	University of Malaga
Jorge Cáncer Gil	University of Zaragoza
Marcos Canales Mayo (team leader)	University of Zaragoza

The team

Borja

University of Cantabria

Carlos

Polytechnic University of Catalonia

Jorge

Marcos

University of Zaragoza

Denisa

University of Malaga

Table of contents

Intro
- Problem description
- Study of existing implementations

Our strategy
- Available platforms
- Implementation decisions

Implementation
- OpenMP
- FPGA + CPU
- (2x)GPU + CPU
- FPGA + CPU + (2x)GPU

Overall results
- Methodology
- Test results
- Conclusions

1.Intro

Problem description
Study of existing implementations

Problem description
Definitions

- **Clustering:** task of assigning a set of objects into groups.
- ***k*-means clustering:**
  - method of clustering
  - partition *n* data points into *k* clusters (n >> k)
  - each point belongs to the cluster with the **nearest** mean.
- The **nearness:** usually Euclidean or Manhattan distance.
- **Assumption:** data points are independent of each other.

Problem description
The algorithm

Input: set of N points with D dimensions
K (number of clusters)

Output: partition of N points in K clusters

1. Place centroids $\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_K}$ at random locations
2. Iterate until convergence condition is met
3. For each point $\boldsymbol{x_i}$, $i=1..N$:
4. For each cluster c_j, $j=1..K$:
5. Get distance to c_j, given all D dimensions
6. Assign membership of $\boldsymbol{x_i}$ to nearest cluster j
7. For each cluster c_j, $j=1..K$:
8. $\boldsymbol{c_j}$ = mean of all points whose membership is j

$$O(\#iterations \times N \times D \times K)$$

Study of existing implementations

Implementation	Approach	Device
1. Rodinia	OpenMP	CPU
	OpenCL	GPU
	CUDA	GPU
2. OpenDwarfs	OpenCL	CPU | GPU FPGA | MIC
3. NU-MineBench	OpenMP	CPU
4. Hetero-Mark	OpenCL	CPU | GPU
5. CyberPoint	MPI	CPUs

Web references:

1. https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/K-Means
2. https://github.com/vtsynergy/OpenDwarfs
3. http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html
4. http://www.ece.neu.edu/groups/nucar/software/hetero-mark/
5. https://github.com/CyberPoint/libem

2.Our strategy

Available platforms
Implementation decisions

Available platforms

- Intel® Core™ i7-6700K

- 2 x NVIDIA TITAN X

- Altera Terasic Stratix V DE5-NET FPGA

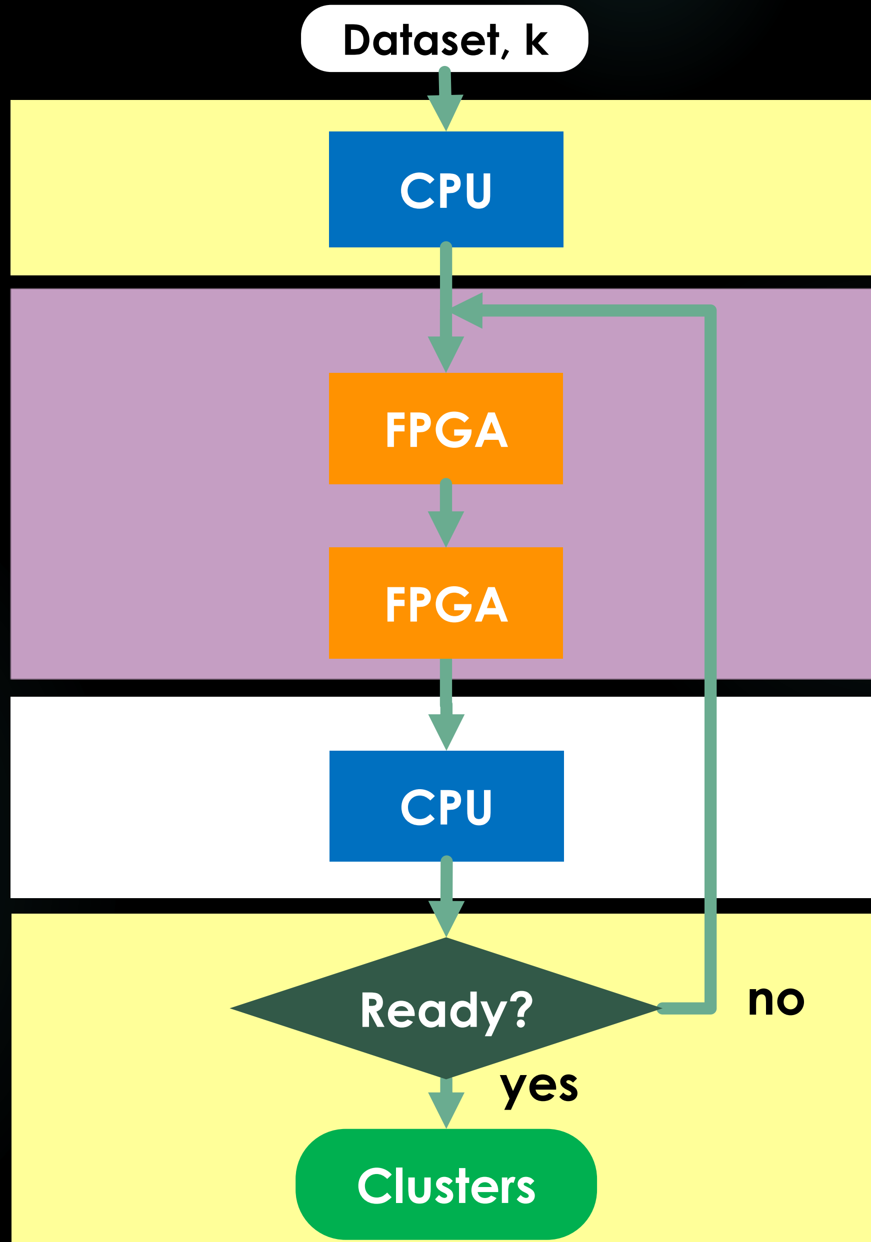
Implementation decisions

- **Considerations:**
  - Points don't change between iterations. They need to be distributed only once among devices.
  - The application is regular
- **Rodinia OpenCL** as a starting point
  - GPU focused.
  - Develop a kernel tuned for the FPGA
- **CPU**
  - Just update centroids and control convergence

3.Implementation

FPGA + CPU
(2x)GPU + CPU
FPGA + CPU + (2x)GPU

FPGA + CPU



Calculate distances

Get membership of each point

Merge & Update clusters centroids

C/C++

OpenCL

OpenMP

FPGA + CPU

SIMD Parallelism (GPU)	1 A	1 B	1 C	1 D	1 E	4 A	4 B	4 C	4 D	4 E
	2 A	2 B	2 C	2 D	2 E	5 A	5 B	5 C	5 D	5 E
	3 A	3 B	3 C	3 D	3 E	6 A	6 B	6 C	6 D	6 E
Clock Cycle	1	2	3	4	5	6	7	8	9	10
Pipeline Parallelism (FPGA)	1 A	2 A	3 A	4 A	5 A	6 A				
		1 B	2 B	3 B	4 B	5 B	6 B			
			1 C	2 C	3 C	4 C	5 C	6 C		
				1 D	2 D	3 D	4 D	5 D	6 D	
					1 E	2 E	3 E	4 E	5 E	6 E

Comparison of SIMD Parallelism Versus Pipeline Parallelism

OpenCL on FPGAs for GPU Programmers, Intel Altera

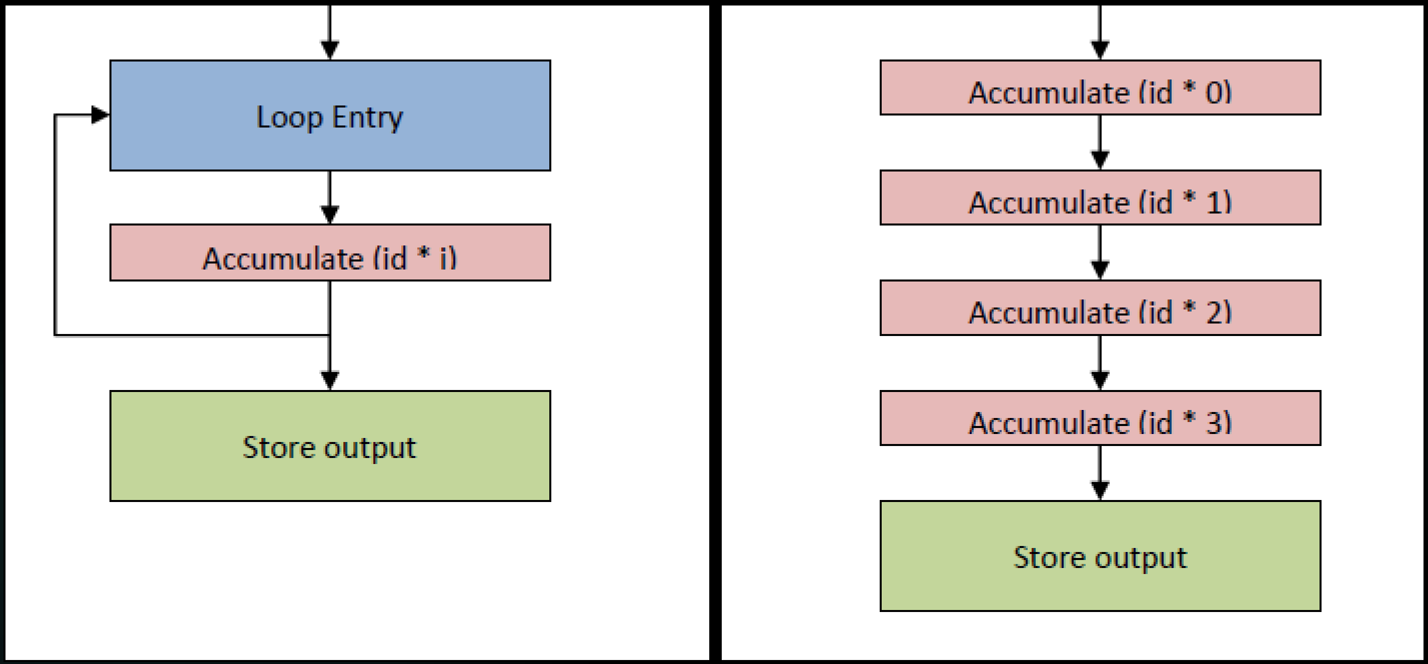
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-201406-acceleware-opencl-on-fpgas-for-gpu-programmers.pdf

FPGA + CPU v0

> 200 s/iter
8192p
D=2
K=2

```
Function Kmeans_Kernel is
    Input:
    pts, clusters, npoints, nclusters, ndims;
    Output:
    membership;
    gid ← get_global_id(0);
    if gid < npoints then
        index ← 0;
        min_dist ← ∞;
        for c ∈ [0, nclusters) do
            dist ← 0;
            for d ∈ [0, ndims) do
                dist += (pts[gid * ndims + d] − clusters[c * ndims + d]) *
                 (pts[gid * ndims + d] − clusters[c * ndims + d]);
            end
            if dist < min_dist then
                min_dist ← dist;
                index ← c;
        end
        membership[gid] ← index;
end
```

FPGA + CPU



Loop Unrolling Example

OpenCL on FPGAs for GPU Programmers, Intel Altera

https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-201406-acceleware-opencl-on-fpgas-for-gpu-programmers.pdf

FPGA + CPU v1

Function *Kmeans_Kernel* **is**

Input:
pts, clusters;

Output:
membership;

```
gid ← get_global_id(0);
if gid < NPOINTS then
    index ← 0;
    min_dist ← ∞;
    for c ∈ [0, NCLUSTERS) do
        dist ← 0;
        #pragma unroll;
        for d ∈ [0, NDIMS) do
            dist += (pts[gid*NDIMS+d] − clusters[c*NDIMS+d]) *
             (pts[gid*NDIMS+d] − clusters[c*NDIMS+d]);
        end
        if dist < min_dist then
            min_dist ← dist;
            index ← c;
    end
    membership[gid] ← index;
end
```

FPGA + CPU v2

Function *Kmeans_Kernel* **is**

Input:

$pts, clusters;$

Output:

$membership;$

$gid \leftarrow get\_global\_id(0);$

if $gid < NPOINTS$ **then**

$index \leftarrow 0;$

$min\_dist \leftarrow \infty;$

for $c \in [0,\ NCLUSTERS)$ **do**

$dist \leftarrow 0;$

#pragma unroll;

for $d \in [0,\ NDIMS)$ **do**

$dist \mathrel{+}= (pts[d * \mathrm{NPOINTS} + gid] - clusters[c * \mathrm{NDIMS} + d]) * (pts[d * \mathrm{NPOINTS} + gid] - clusters[c * \mathrm{NDIMS} + d]);$

end

if $dist < min\_dist$ **then**

$min\_dist \leftarrow dist;$

$index \leftarrow c;$

end

$membership[gid] \leftarrow index;$

end

FPGA + CPU v3

```
Function Kmeans_Kernel (Workgroup size = NCLUSTERS*NDIMS) is
    Input:
    pts, clusters;
    Output:
    membership;
    gid ← get_global_id(0);
    lid ← get_local_id(0);
    clusters_local[NCLUSTERS*NDIMS];
    clusters_local[lid] ← clusters[lid];
    barrier(CLK_LOCAL_MEM_FENCE);
    index ← 0;
    min_dist ← ∞;
    for c ∈ [0, NCLUSTERS) do
        dist ← 0;
        #pragma unroll
        for d ∈ [0, NDIMS) do
            dist +=
             (pts[d*NPOINTS+gid] − clusters_local[c*NDIMS+d]) *
             (pts[d*NPOINTS+gid] − clusters_local[c*NDIMS+d]);
        end
        if dist < min_dist then
            min_dist ← dist;
            index ← c;
    end
    membership[gid] ← index;
end
```

FPGA + CPU v4

0.34 s/iter
8192p
D=2
K=2

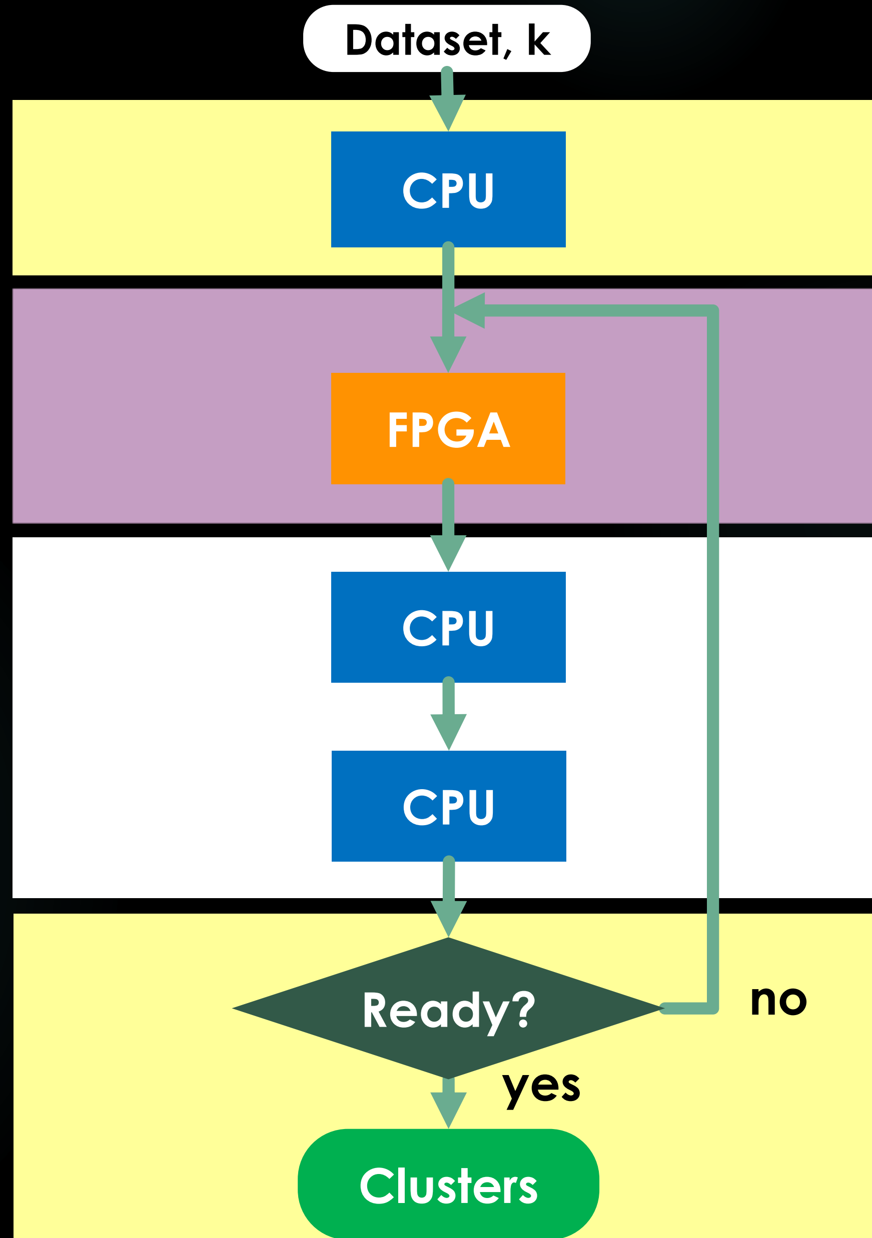
```
Function Kmeans_Kernel (Workgroup size = NCLUSTERS*NDIMS) is
    Input:
    feature, clusters;
    Output:
    distances;
    gid ← get_global_id(0);
    lid ← get_local_id(0);
    clusters_local[NCLUSTERS*NDIMS];
    clusters_local[lid] ← clusters[lid];
    barrier(CLK_LOCAL_MEM_FENCE);
    index ← 0;
    min_dist ← ∞;
    #pragma unroll 4
    for c ∈ [0, NCLUSTERS) do
        dist ← 0;
        #pragma unroll
        for d ∈ [0, NDIMS) do
            diff =
             feature[d*NPOINTS+gid] − clusters_local[c*NDIMS+d]);
            dist += pown(diff, 2);
        end
        distances[gid*NCLUSTERS+c] ← dist;
    end
end
```

FPGA + CPU

Calculate
distances

Get membership
of each point

Merge & Update
clusters centroids



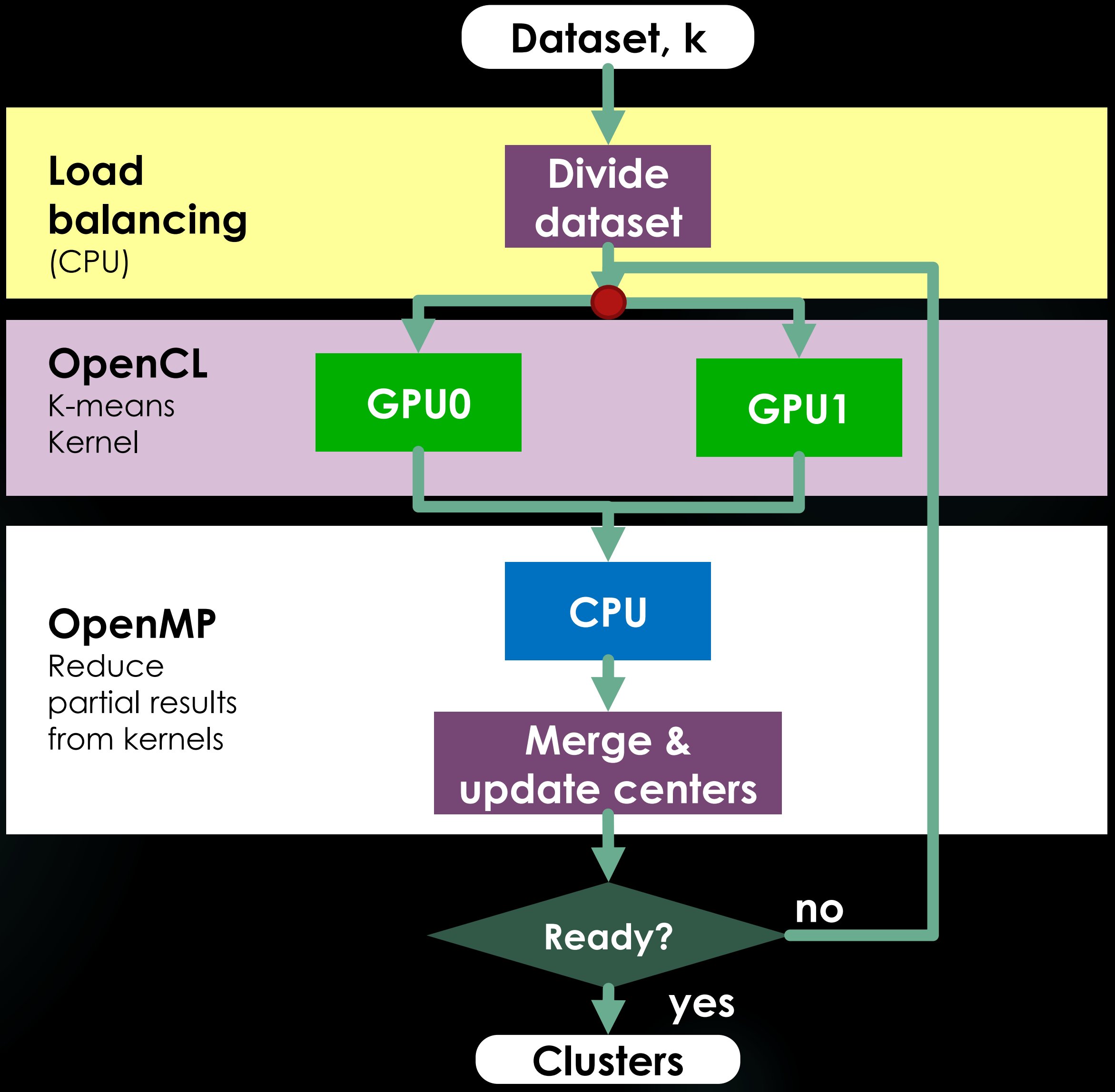
C/C++

OpenCL

OpenMP

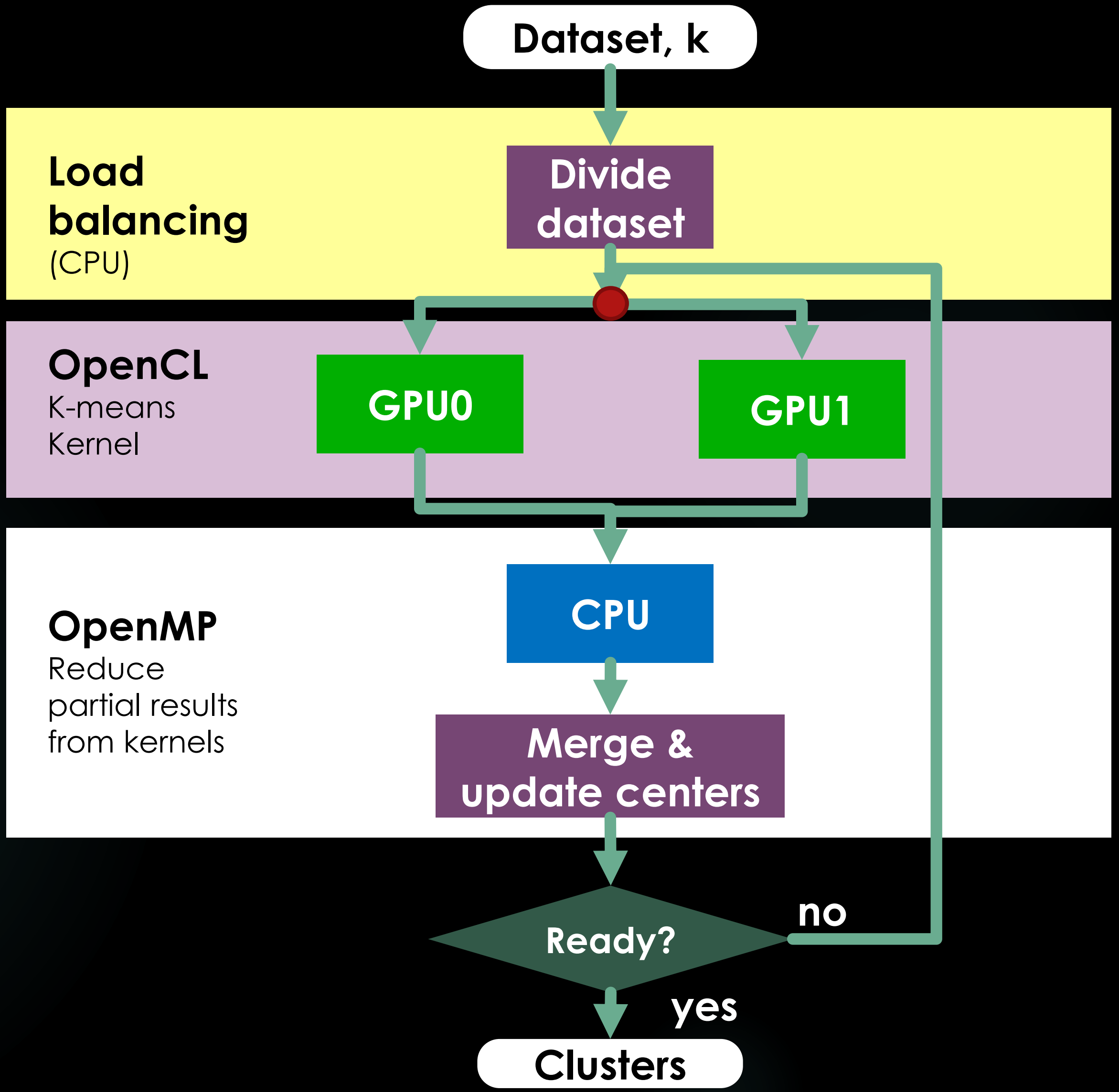
(2x)GPU + CPU

- No need to adapt the kernel
- Focus on orchestrating both devices



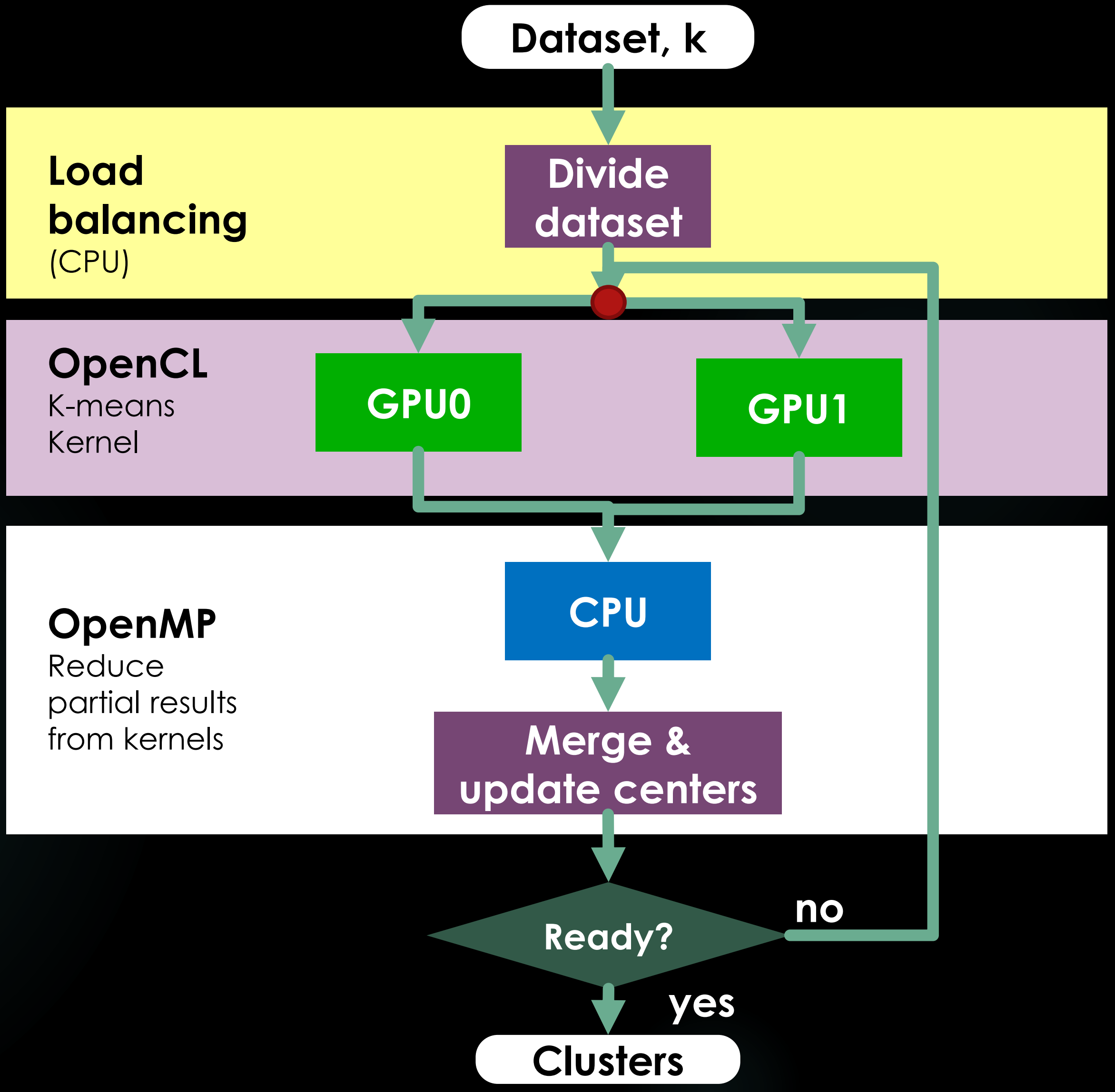
(2x)GPU + CPU

- OpenCL buffer sharing causes performance degradation



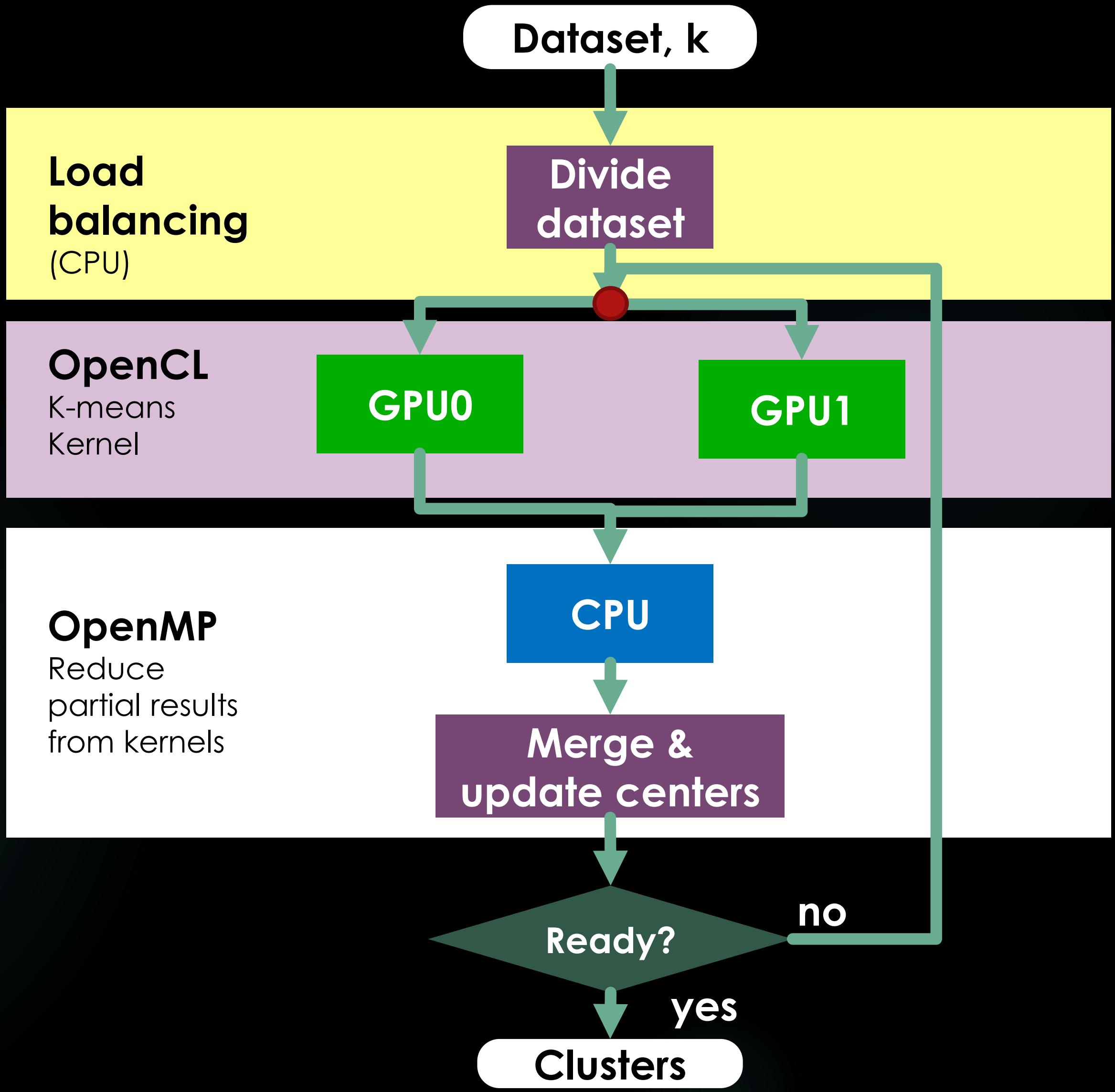
(2x)GPU + CPU

- To preserve coalescing, input points must be stored as a SoA
  - Buffers must be swapped



(2x)GPU + CPU

- Both devices are identical, so the load can be evenly distributed



FPGA + CPU + (2x)GPU

Split work for all devices

Calculate distances

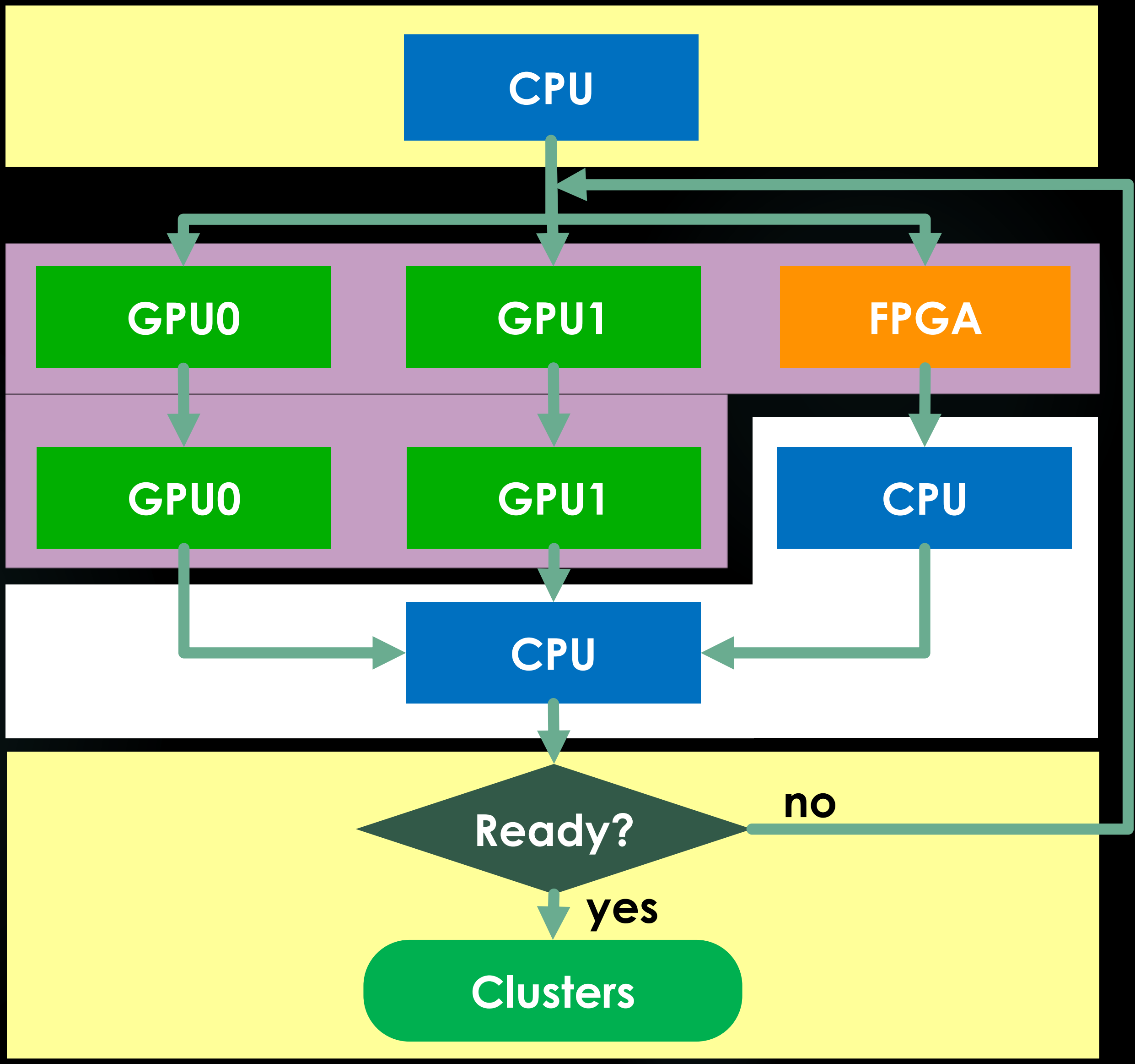
Get membership of each point

Merge & Update clusters centroids

C/C++

OpenCL

OpenMP



4.Overall results

Methodology
Test results

Methodology

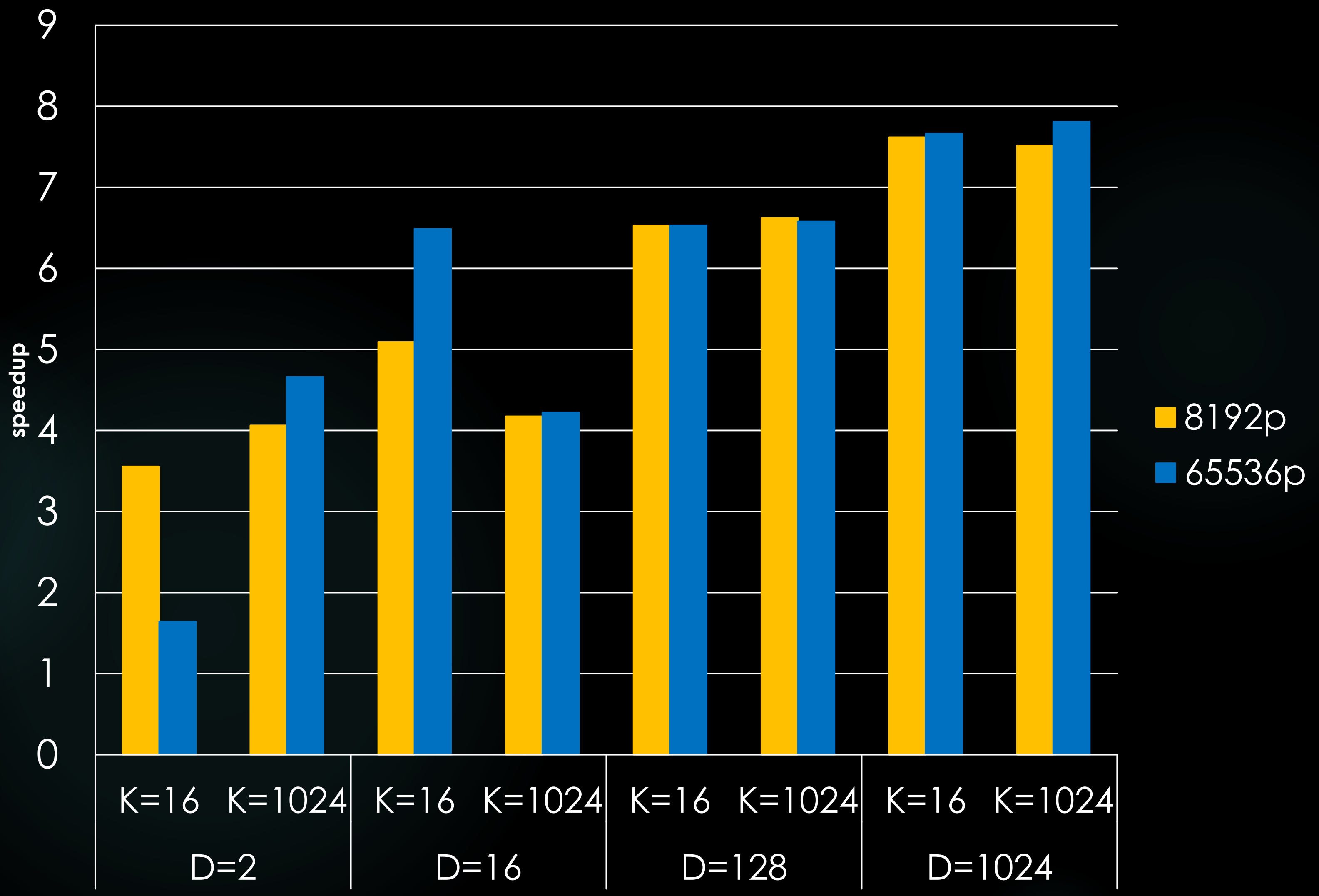
Compare execution time:

- Base system: Sequential
- OpenMP (8 threads)
- 1 GPU
- 2 GPUs
- FPGA
- CPU-(2x)GPU-FPGA

Datasets used:

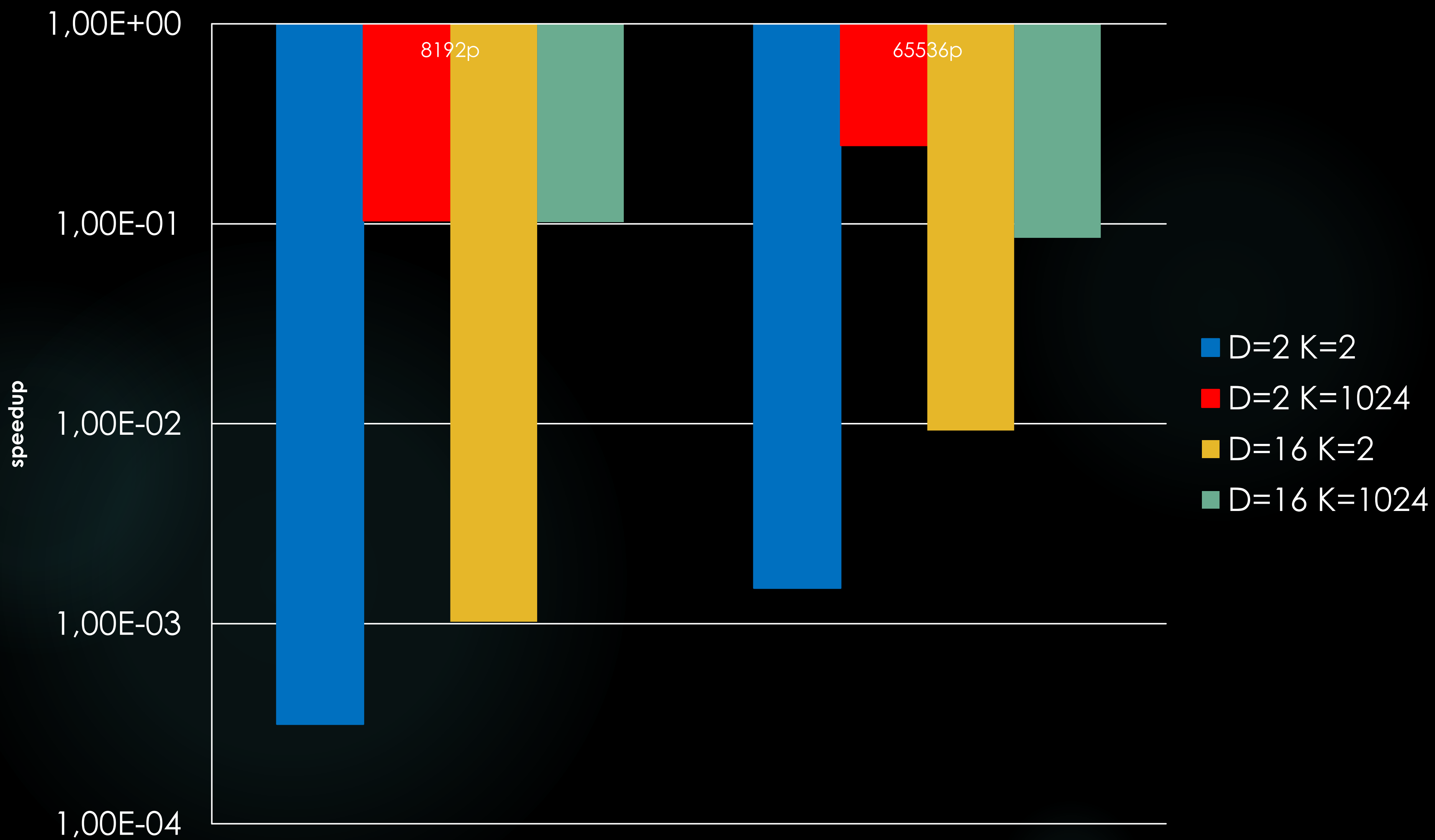
- 8192 points
  - 2 dimensions
  - 16 dimensions
  - 128 dimensions
  - 1024 dimensions
- 65536 points
  - 2 dimensions
  - 16 dimensions
  - 128 dimensions
  - 1024 dimensions
- 4194304 points
  - 2 dimensions
  - 16 dimensions
  - 128 dimensions
  - 1024 dimensions

OpenMP



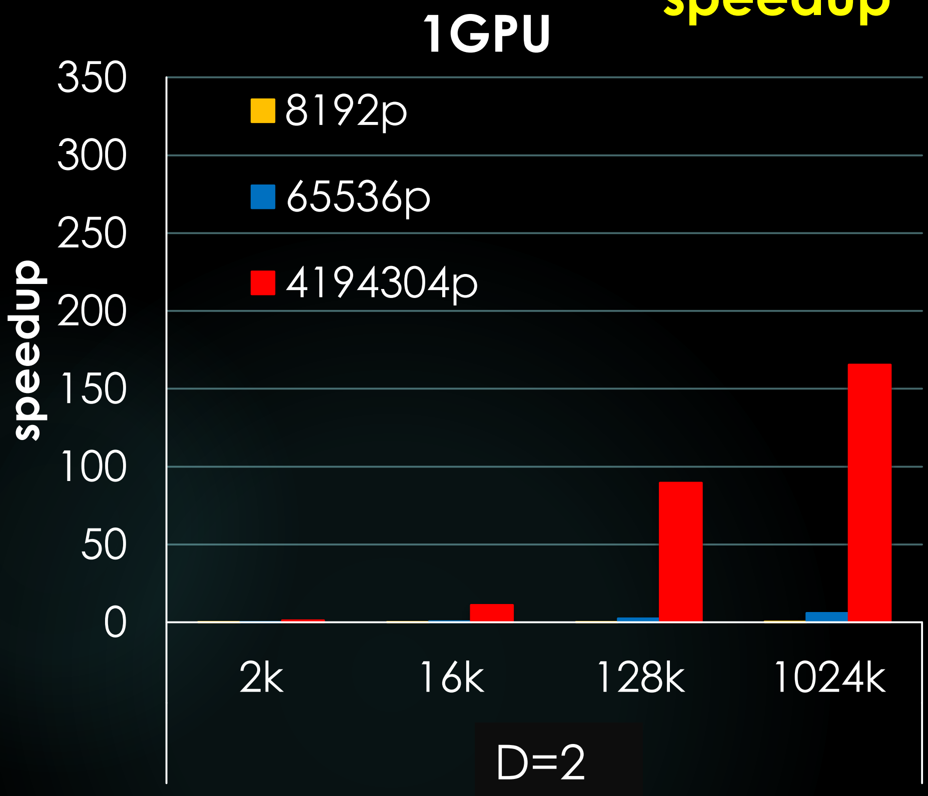
speedup
0
1
2
3
4
5
6
7
8
9
K=16
K=1024
D=2
D=16
D=128
D=1024
8192p
65536p

FPGA + CPU

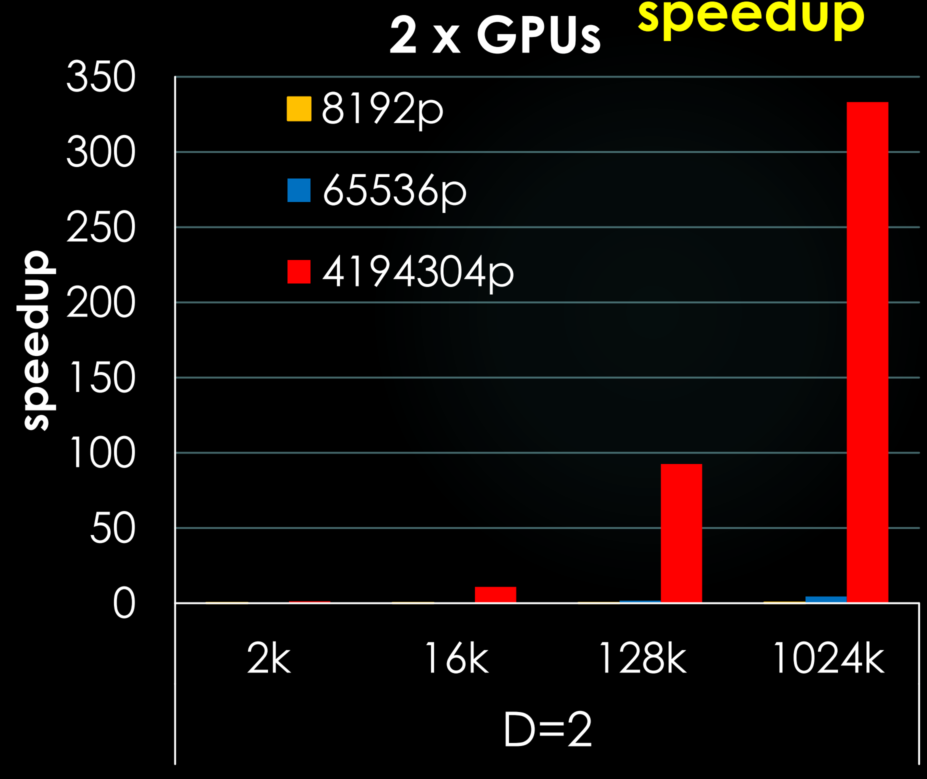

1,00E+00
1,00E-01
1,00E-02
1,00E-03
1,00E-04
speedup
8192p
65536p
D=2 K=2
D=2 K=1024
D=16 K=2
D=16 K=1024

(2x)GPU + CPU

164 x speedup

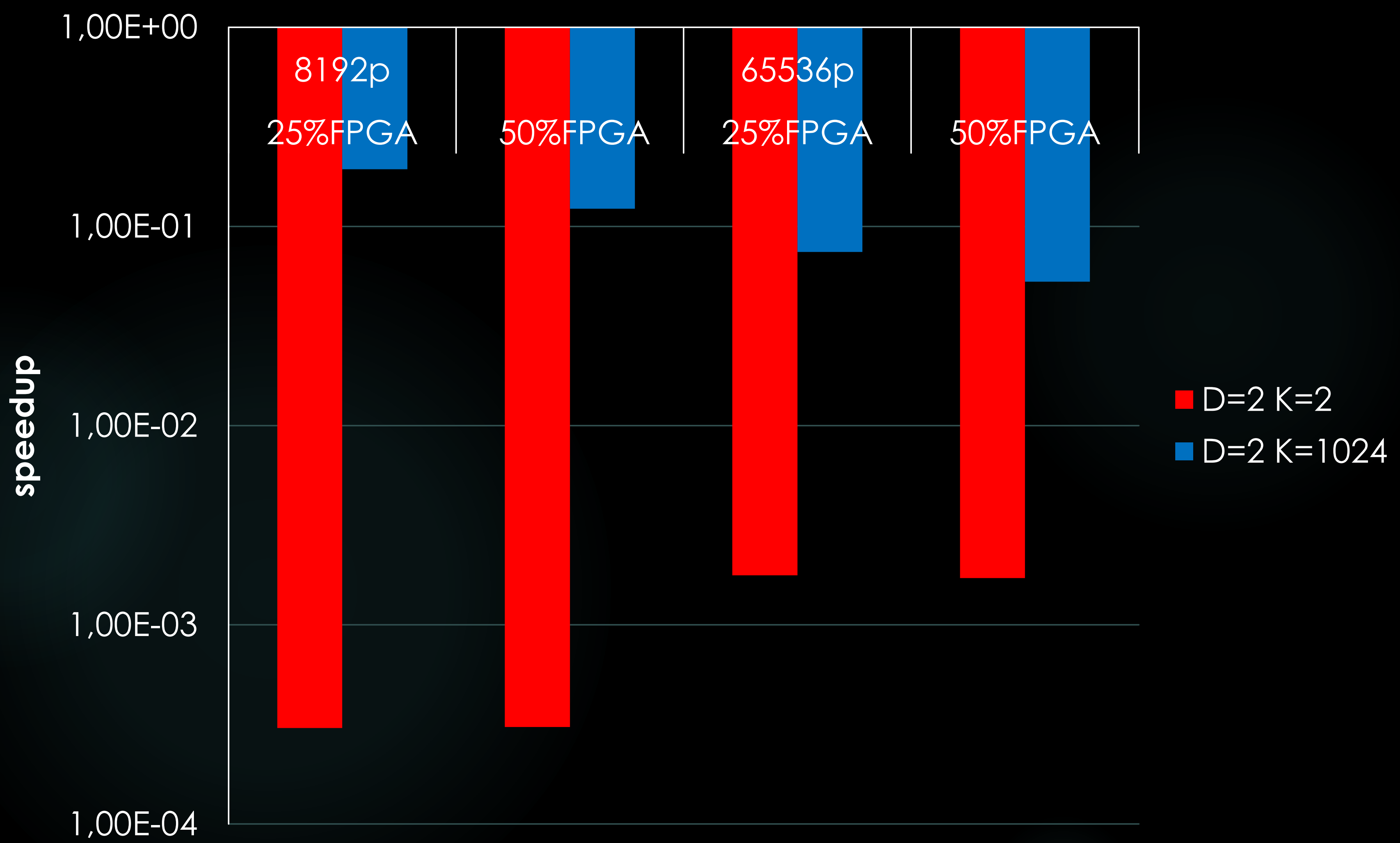


328 x speedup

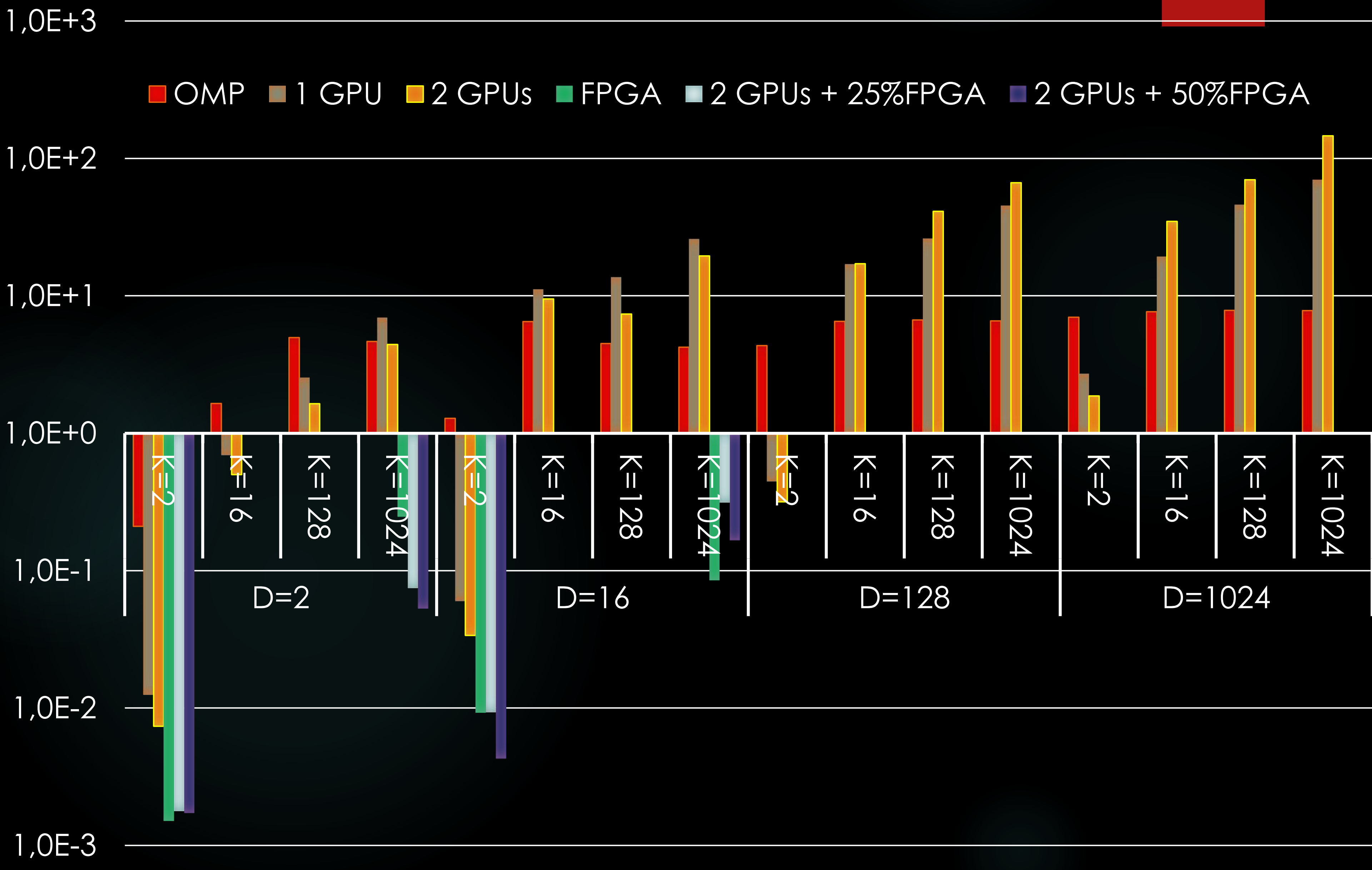


- The 2 GPU implementation requires large problem sizes for the distribution to be worthy

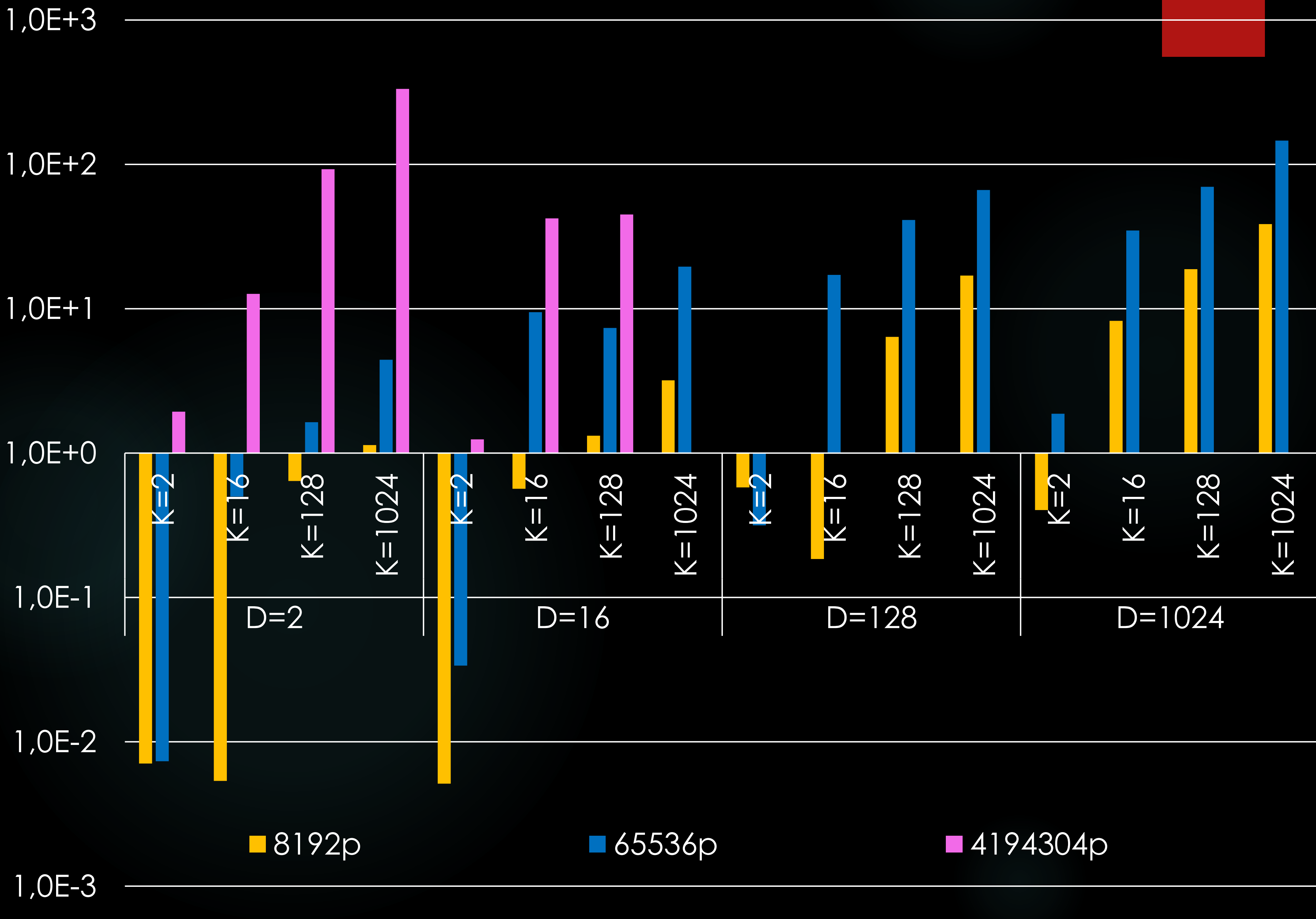
FPGA + CPU + (2x)GPU



Speedup comparison Npoints = 65536



Npoints 2-GPUS Speedup Comparison



5.Conclusions

Conclusions

- Small datasets => **OMP** is the best choice (delegating on other devices is not worth it – high I/O time).
- Large datasets => **GPUs** (massive parallelism). **328 x speedup!**
- Difficult to get advantage of **FPGA's** resources by pipelining the k-means algorithm.
- **Integrated GPU & FPGA** version to further analyze the tradeoff between the overall execution time and the power usage.
- **Dynamic load balancer** - split the workload depending on specific criteria (e.g. execution time of previous iterations for each device).

$$O(\#iterations \times N \times D \times K)$$

References

1. Tang, Q. Y., & Khalid, M. A. (2016). Acceleration of K-Means Algorithm Using Altera SDK for OpenCL. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 10(1), 6.
2. K-Means. Rodinia. Retrieved April 23, 2017, from www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/K-Means
3. OpenMP. Retrieved April 23, 2017, from www.openmp.org/
4. Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. Retrieved April 23, 2017, from www.khronos.org/opencl/
5. Intel. Intel FPGA SDK for OpenCL. Retrieved April 23, 2017, from www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf

Source Code

- https://github.com/MarcosCM/Heterogeniuses