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
University of Zaragoza

Marcos

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 ($n \gg k$)
  - 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 \( c_1, c_2, \ldots, c_K \) at random locations

2. Iterate until convergence condition is met

3. For each point \( x_i, i=1..N \):

4. For each cluster \( c_j, j=1..K \):

5. Get distance to \( c_j \), given all \( D \) dimensions

6. Assign membership of \( x_i \) to nearest cluster \( j \)

7. For each cluster \( c_j, j=1..K \):

8. \( c_j = \text{mean of all points whose membership is} \ j \)

\( O(\text{#iterations} \times N \times D \times K) \)
## Study of existing implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Approach</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Rodinia</td>
<td>OpenMP</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>OpenCL</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>CUDA</td>
<td>GPU</td>
</tr>
<tr>
<td>2. OpenDwarfs</td>
<td>OpenCL</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FPGA</td>
</tr>
<tr>
<td>3. NU-MineBench</td>
<td>OpenMP</td>
<td>CPU</td>
</tr>
<tr>
<td>4. Hetero-Mark</td>
<td>OpenCL</td>
<td>CPU</td>
</tr>
<tr>
<td>5. CyberPoint</td>
<td>MPI</td>
<td>CPUs</td>
</tr>
</tbody>
</table>

### Web references:

2. [https://github.com/vtsynergy/OpenDwarfs](https://github.com/vtsynergy/OpenDwarfs)
3. [http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html](http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html)
4. [http://www.ece.neu.edu/groups/nucar/software/hetero-mark/](http://www.ece.neu.edu/groups/nucar/software/hetero-mark/)
5. [https://github.com/CyberPoint/libem](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

Dataset, k

CPU

FPGA

FPGA

CPU

Ready? no

yes

Clusters

C/C++

OpenCL

OpenMP
### Comparison of SIMD Parallelism Versus Pipeline Parallelism

OpenCL on FPGAs for GPU Programmers, Intel Altera


<table>
<thead>
<tr>
<th>SIMD Parallelism</th>
<th>(GPU)</th>
<th>Clock Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 A</td>
<td>1 B</td>
<td>1 C</td>
<td>1 D</td>
<td>1 E</td>
<td>4 A</td>
<td>4 B</td>
<td>4 C</td>
<td>4 D</td>
<td>4 E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 A</td>
<td>2 B</td>
<td>2 C</td>
<td></td>
<td></td>
<td></td>
<td>5 A</td>
<td>5 B</td>
<td>5 C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 A</td>
<td>3 B</td>
<td>3 C</td>
<td></td>
<td></td>
<td></td>
<td>6 A</td>
<td>6 B</td>
<td>6 C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 A</td>
<td>2 A</td>
<td>3 A</td>
<td>4 A</td>
<td>5 A</td>
<td>6 A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 B</td>
<td>2 B</td>
<td>3 B</td>
<td>4 B</td>
<td>5 B</td>
<td>6 A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 C</td>
<td>2 C</td>
<td>3 C</td>
<td>4 C</td>
<td>5 C</td>
<td>6 C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 D</td>
<td>2 D</td>
<td>3 D</td>
<td>4 D</td>
<td>5 D</td>
<td>6 D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 E</td>
<td>2 E</td>
<td>3 E</td>
<td>4 E</td>
<td>5 E</td>
<td>6 E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FPGA + CPU**
FPGA + CPU v0

Function $Kmeans\_Kernel$ is

Input:
$pts, clusters, npoints, nclusters, ndims$;

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
    $dist \leftarrow 0;$
    for $d \in [0, ndims)$ do
      $dist += (pts[gid * ndims + d] - clusters[c * ndims + d]) *$
      $(pts[gid * ndims + d] - clusters[c * ndims + d]);$
    end
    if $dist < min\_dist$ then
      $min\_dist \leftarrow dist;$
      $index \leftarrow c;$
    end
  end
  $membership[gid] \leftarrow index;$
end

$> 200 \text{ s/iter}$

8192p

D=2

K=2
FPGA + CPU

Loop Unrolling Example
OpenCL on FPGAs for GPU Programmers, Intel Altera
Function $K$means $Kernel$ is

Input:
$pts, clusters$;

Output:
$membership$;

$gid \leftarrow \text{get\_global\_id}(0);$ 
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 $+= (pts[gid\ast NDIMS + d] - clusters[c\ast NDIMS + d]) *$
    $(pts[gid\ast NDIMS + d] - clusters[c\ast NDIMS + d]);$
end

if dist $< min\_dist$ then

    min_dist $\leftarrow$ dist;
    index $\leftarrow$ c;
end

membership[gid] $\leftarrow$ index;
end
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[d*NPOINTS+gid] − clusters[c*NDIMS+d]) * 
                        (pts[d*NPOINTS+gid] − clusters[c*NDIMS+d]);
            end
            if dist < min_dist then
                min_dist ← dist;
                index ← c;
            end
        end
        membership[gid] ← index;
    end
FPGA + CPU v3

Function Kmeans_Kernel (Workgroup size = NCLUSTERS*NDIMS) is

Input:
pts, clusters;

Output:
membership;

$\text{gid} \leftarrow \text{get\_global\_id}(0)$;
$\text{lid} \leftarrow \text{get\_local\_id}(0)$;

clusters.local[NCLUSTERS*NDIMS];
clusters.local[\text{lid}] \leftarrow \text{clusters[lid]}$

\text{barrier}($\text{CLK\_LOCAL\_MEM\_FENCE}$);

index \leftarrow 0;
min\_dist \leftarrow \infty;

for $c \in [0, \text{NCLUSTERS})$ do

$$\text{dist} \leftarrow 0;$$

#pragma unroll

for $d \in [0, \text{NDIMS})$ do

$$\text{dist} + =$$

\quad (\text{pts}[\text{d}\times\text{NPOINTS}\text{+}\text{gid}] - \text{clusters.local}[c\times\text{NDIMS}\text{+}\text{d}]) \times$

\quad (\text{pts}[\text{d}\times\text{NPOINTS}\text{+}\text{gid}] - \text{clusters.local}[c\times\text{NDIMS}\text{+}\text{d}]);

end

if $\text{dist} < \text{min\_dist}$ then

$$\text{min\_dist} \leftarrow \text{dist};$$

$$\text{index} \leftarrow c;$$

end

membership[\text{gid}] \leftarrow \text{index};

end
FPGA + CPU v4

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

0.34 s/iter
8192p
D=2
K=2
FPGA + CPU

Calculate distances

Get membership of each point

Merge & Update clusters centroids

Dataset, k

CPU

FPGA

CPU

CPU

Ready?

no

yes

Clusters

C/C++

OpenCL

OpenMP
(2x)GPU + CPU

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

Load balancing (CPU)

Dataset, k

OpenCL
K-means Kernel

GPU0

GPU1

OpenMP
Reduce partial results from kernels

CPU

Merge & update centers

Ready?

yes

no

Clusters
(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

Load balancing (CPU)

Divide dataset

OpenCL
K-means Kernel

GPU0

GPU1

OpenMP
Reduce partial results from kernels

CPU

Merge & update centers

Ready?

yes

no

Clusters
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

CPU

GPU0

GPU1

FPGA

Get membership of each point

CPU

GPU0

GPU1

CPU

Ready?

no

yes

Clusters
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

[Graph showing speedup for different configurations of K and D, with two sets of data points indicated by 8192p and 65536p.]
The 2 GPU implementation requires large problem sizes for the distribution to be worthy.
FPGA + CPU + (2x)GPU

![Bar chart showing speedup for different configurations.

- D=2 K=2
- D=2 K=1024

Configuration details:
- 8192p
- 65536p
- 50% FPGA
- 25% FPGA

Speedup values range from 1.00E-04 to 1.00E+00.]
Speedup comparison Npoints = 65536
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).

\[ O(\#\text{iterations} \times N \times D \times K) \]
References


Source Code

https://github.com/MarcosCM/Heterogeniuses